NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

A SOFTWARE RELIABILITY ENGINEERING CASE STUDY

by

Judie A. Heineman

March, 1996

Thesis Advisor: Norman F. Schneidewind

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March, 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE A SOFTWARE RELIABILITY ENGINEERING CASE STUDY			5. FUNDING NUMBERS
6. AUTHOR(S) Heineman, Judie A.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE
13. ABSTRACT *(maximum 200 words)* Handling, identifying, and correcting faults are significant concerns for the software manager because (1) the presence of faults in the operational software can put human life and mission success at risk in a safety critical application and (2) the entire software reliability process is expensive. Designing an effective Software Reliability Engineering (SRE) process is one method to increase reliability and reduce costs. This thesis describes a process that is being implemented at Marine Corps Tactical System Support Activity (MCTSSA), using the Schneidewind Reliability Model and the SRE process described in the American Institute of Aeronautics and Astronautics Recommended Practice in Software Reliability. In addition to applying the SRE process to single node systems, its applicability to multi-node LAN-based distributed systems is explored. Each of the SRE steps is discussed, with practical examples provided, as they would apply to a testing facility. Special attention is directed to data collection methodologies and the application of model results. In addition, a handbook and training plan are provided for use by MCTSSA during the transition to the SRE process.			
14. SUBJECT TERMS Software Reliability Engineering Process, Reliability Modeling, Multi-node Reliability Model			15. NUMBER OF PAGES 182
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

A SOFTWARE RELIABILITY ENGINEERING CASE STUDY

Judie A. Heineman
Lieutenant Commander, United States Navy
B.A., College of the Holy Cross, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1996

Author: J Heineman
Judie A. Heineman

Approved by: Norman F. Schneidewind
Norman F. Schneidewind, Thesis Advisor

[illegible]
James Emery, Associate Advisor

Reuben T Harris
Reuben T. Harris, Chairman
Department of Systems Management

ABSTRACT

Handling, identifying, and correcting faults are significant concerns for the software manager because (1) the presence of faults in the operational software can put human life and mission success at risk in a safety critical application and (2) the entire software reliability process is expensive. Designing an effective Software Reliability Engineering (SRE) process is one method to increase reliability and reduce costs. This thesis describes a process that is being implemented at Marine Corps Tactical System Support Activity (MCTSSA), using the Schneidewind Reliability Model and the SRE process described in the American Institute of Aeronautics and Astronautics Recommended Practice in Software Reliability. In addition to applying the SRE process to single node systems, its applicability to multi-node LAN-based distributed systems is explored. Each of the SRE steps is discussed, with practical examples provided, as they would apply to a testing facility. Special attention is directed to data collection methodologies and the application of model results. In addition, a handbook and training plan are provided for use by MCTSSA during the transition to the SRE process.

TABLE OF CONTENTS

I. SOFTWARE RELIABILITY

A. HARDWARE VS. SOFTWARE RELIABILITY

While most people are familiar with hardware reliability , software reliability appears to be a concept that needs some clarification. Contrasting the two concepts may shed some light on the distinctions. The American Institute of Aeronautical Engineers (AIEE, 1993) provides the following examples:

- Changes to hardware systems are extensive and time-consuming due to the physical nature of hardware. Changing software is frequently more feasible because software can be easily changed with a text editor. For example, it can be adapted to changing user requirements, whereas this would not be feasible with hardware. However, this software flexibility has caused great problems in the industry because of inadequate specifications, testing, and maintenance of software changes.

- Software has no physical existence. Since it includes both data and logic, any item can be a source of failure.

- Failures attributable to software faults frequently occur without advance warning.

- Repair generally restores hardware to its previous state. Correction of software problems always changes the software to a state different from the one prior to the change.

- Redundancy and fault tolerance for hardware are common practice. These concepts are only beginning to be applied to software.

- A high rate of software changes can be detrimental to software reliability.

With these distinctions in mind, one can begin to understand the challenges an organization faces when it deals with software and its reliability. Not only are most personnel unfamiliar with the concept of software reliability, they are certainly challenged when it comes to *predicting* its reliability.

B. DEFINITION OF SOFTWARE RELIABILITY

Reliability is seen as the ability of a system to perform as expected under specific conditions for a specified period of time. This also includes the "**probability** of failure-free operation of a computer program for a specified time in a specified environment." (Musa, 1987) The challenge occurs when this concept must be matched with appropriate measurement techniques to evaluate the software's ability to perform.

C. USES FOR THE SOFTWARE RELIABILITY ENGINEERING (SRE) PROCESS

Software Reliability Engineering (SRE) is a new discipline that is maturing as more organizations see the need to develop standard reliability practices. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as "the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems." (AIAA, 1993) This formalized process helps prevent organizations from adjusting and modifying software "on the fly" and encourages an engineering way of conducting business. This methodology is especially helpful for the software manager and the user. Musa (1987) proposes four specific ways in which software reliability measures can be of great value to the manager and user.

First, the reliability measures provide a means of quantitatively evaluating the software. Organizations are frequently introducing new techniques for improving the means by which software is designed. However, these techniques do not include any methodology for distinguishing between good and bad new technology. Software reliability measures offer the promise of providing at least one criterion for evaluating the new technology. As an example, the organization can compare the number of failures per unit time of the new technology versus the old. (Musa, 1987)

Second, a software reliability measure permits the user to evaluate the development status of the product during the test phases of the project. In the past, evaluation criteria used were purely subjective: intuition of the designer, percentage of tests completed, and successful completion of a specific number of tests. An objective reliability measure, such as the failure intensity mentioned above, can provide a sound means for evaluating development status. (Musa, 1987) Additionally, the measurement of residual faults and failures is gaining in popularity and provides a more intuitive understanding of the status of the software's reliability. Residual faults and failures address the issue of remaining problems in the software and provide a means of quantifying the risk of experiencing a software failure during software execution. (Keller, 1995) They also provide a means of "rationalizing how long to test a piece of software." Having predictions regarding the extent to which the software is *not* fault free is meaningful for assessing the risk of deploying the software. (Schneidewind, 1996)

Third, software reliability measures can be used to monitor the operational performance of the software and evaluate any changes made during design. Typically, as more changes are made to software, its ability to perform as expected (reliability) decreases. (Musa, 1987)

Finally, the manager and user are given a better insight into the various factors affecting and influencing software reliability. This provides them with the capability of making much more informed decisions. (Musa, 1987)

D. APPLICABILITY OF THE SRE PROCESS TO SOFTWARE MANAGERS

Handling, identifying and correcting faults is a significant concern for the manager because the entire software reliability process is expensive. "It also impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time and peripherals requirements)." (AIAA, 1993) This addresses the key issue regarding SRE -- *it provides the manager with information about which he can make informed decisions.*

There will always be a tradeoff between reliability, sometimes referred to as the failure rate, and cost. (Cost is directly related to testing time). The manager will need to decide on a certain level of reliability for the product, resulting in a set cost. Thus, higher reliability will result in a higher cost. The converse is also true.

In general, the failure rate of a software system is seen as a curve with a decreasing slope which results from the identification and removal of faults as time passes. It is the primary purpose of reliability modeling to define the shape of this resulting curve using statistical methodologies. The model used in these reliability assessments can provide prediction information regarding the software execution time needed to discover a specified number of faults, or predict the time period when the next fault will occur. Figure 1 provides a sample software reliability curve that can be generated by using the results of a software reliability model. (AIAA, 1993)

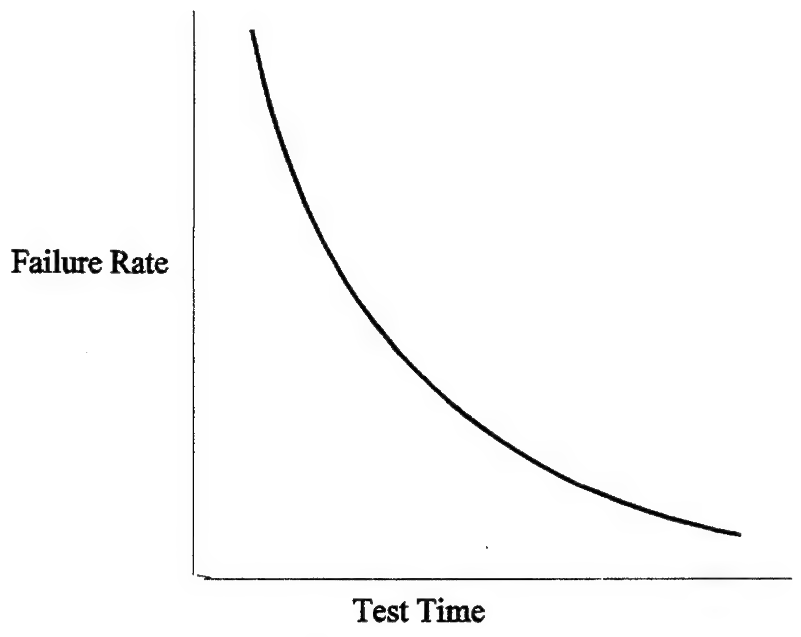


Figure 1: Software Reliability Tradeoff Curve

E. THESIS OBJECTIVES AND PURPOSE

Of interest in this thesis is the applicability of the SRE process to DOD organizations. This thesis will further discuss and evaluate the use of the SRE process at the Marine Corps Tactical Systems Support Activity (MCTSSA), Camp Pendleton, CA. Specifically, it will discuss the generic, recommended steps in implementing an SRE program and will further expand this discussion to include application of the concepts to actual MCTSSA data obtained from a current project, the Marine Air-Ground Task Force II/Logistics Automated Information Systems (or LOGAIS, for short). The result of this study will produce a design for a distributed systems SRE process and subsequent training program. This training program addresses AIAA and IEEE software measurement standards and concepts relating to an SRE program and its implementation.

II. THE SRE PROCESS

A. SRE PROCESS DISCUSSION

As previously mentioned, the software reliability engineering process allows managers to quantitatively evaluate the software delivered to them. It provides for management of risk by predicting the number of faults in the code and the probability of encountering those faults during software execution. It permits the manager to assess the current status of a project by forecasting the reliability of the software and can be used as a metric for process improvement evaluation, for comparing competing products, and for safety certification. Additionally, it permits managers to plan for scheduled introduction of new components and plan for proper resource allocation. (Stark, 1992)

"The primary benefit of an SRE process, however, is that it permits a customer-oriented measure of quality." (Stark, 1992) A common ground is provided for discussion among the engineer, the manager, and the user. Key here is the fact that software reliability can be used throughout the life-cycle to make trade-offs between cost, schedule, and quality. (Stark, 1992)

B. BASIC CONCEPTS

Each discipline uses terminology specific to its domain. The same is true for the SRE process. With this in mind, the following definitions should provide a common baseline for further discussion of the SRE process.

As with any intellectual product, errors in design may occur. An error can be defined as "a discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition." (AIAA, 1993) In software, these errors may appear while completing requirements formulation or, as is often the case, during design, coding, and testing the product. The software development process should include measures to discover and correct faults resulting from these errors. [In this context, faults are defined as "defects in the code that can be the cause of one or more failures." (AIAA, 1993)]

These measures can address reviews, audits, screening by language-dependent tools, and several layers of testing. One way to reduce the number and criticality of faults is by modeling the effects of the remaining faults in the delivered product. This can be achieved through a dedicated measurement process by which each defect or fault is noted and formally recorded for inclusion in the reliability model. (AIAA, 1993)

As a point of clarification, a fault is technically different from a failure. A failure can be defined as "the inability of a system or system component to perform a required function within specified limits" or the "departure of program operation from program requirements." (AIAA, 1993) In simpler terms, a fault usually leads to a failure.

C. COMPONENTS OF AN SRE PROGRAM

A successful software reliability program consists of more than just a model. It also consists of the support structure: reliability requirements; reliability measurements to meet those requirements; data collection procedures to obtain the necessary data; definition of severity levels of failures; applications of reliability predictions; interpretation of model predictions; and user feedback for model improvements. (Schneidewind, 1995) Although the conceptualization of the model does not occur in a sequence of steps as mentioned above, its *implementation* does. The practitioner can best understand this process from a description of the chronology of implementing and applying the model. Therefore, this approach will be used in explaining the SRE process and the application of the selected reliability model. (Schneidewind, 1995)

The SRE methodology used for the MCTSSA project is based on the *Schneidewind Software Reliability Model* (Schneidwind, 1993; Schneidewind, 1975), one of the four models recommended in the *AIAA Recommended Practice for Software Reliability* (AIAA, 1993). The validation is based on the fact the model is used to *assist* in assessing the reliability of the NASA Space Shuttle flight software. According to Ted Keller, Manager, Project Coordination, Onboard Shuttle Software Systems, Loral Space Information Systems: "The *Shuttle* software project is experimenting with a promising algorithm which involves the use of the *Schneidewind Software Reliability Model* to compute a parameter: *fraction of remaining failures* as a function of the archived failure history during testing and operation" (Keller, 1995). Remaining failures, fraction of remaining failures, and time to next failure would not be used to the exclusion of other approaches in making reliability assessments. These metrics would be combined with process procedures such as inspections, defect prevention, project control boards, process assessment, and fault tracking, to provide a quantitative basis for achieving reliability objectives. (Billings, 1994; Schneidewind, 1995)

The standard practices described under the Generic Steps for Implementing an SRE Program are essentially those recommended in the *AIAA Recommended Practice for Software Reliability* (AIAA, 1993) and the *ANSI/IEEE Standard for a Software Quality Metrics Methodology* (IEEE, 1993).

D. GENERIC STEPS FOR IMPLEMENTING AN SRE PROGRAM

Implementing a software reliability program is a two-phased process. It consists of (1) identifying the reliability goals and (2) testing the software to see how it conforms to the stated objectives. The reliability goals can be ideal or conceptual, e.g., zero defects, but should have some basis in reality. The testing phase is the most complex since it involves the actual collection of raw defect data and molding the data to fit the selected model.

With these phases being the stated objective, the following steps should be considered by the organization as it begins to develop a software reliability program. These steps provide a "cookbook" approach to the SRE process and are ordinarily followed sequentially. Each step will be discussed briefly to provide a general understanding of its purpose. Stages that require numerical calculations and application of specific model parameters will be noted. The AAIA (1993) SRE steps are:

- State the Reliability Requirement
- Establish a Measurement Framework
- Collect the Data
- Establish Problem Severity Levels
- Estimate Model Parameters
- Select the Optimal Set of Failure Data
- Identify the Operational Profile
- Make Reliability Predictions
- Validate the Model
- Make Reliability Decisions
- Use Software Reliability Tools

1. Stating the Reliability Requirement

In this step, the software manager should describe the condition that must be fulfilled for the software to be considered satisfactory (reliable). In some cases the reliability specification can be quantified: no more than one critical software failure (i. e., causes the system crash) in an ATM machine per 10,000 hours of operation. In other cases, the specification is stated qualitatively: "The product will have no software failure that would result in loss of life, loss of mission, or cancellation of mission."

2. Establishing a Measurement Framework

One approach the organization could employ would be to take the software from the developer at delivery and run it on its own systems and see how well, or poorly, the software performed. However, if the manager adopted this approach, many months could be wasted if the software is deemed unreliable after post-delivery testing. A better way would be to have some indications of the system's reliability before the software is delivered to the organization, specifically, by conducting developmental or operational testing.

DOD, and other organizations that develop or procure software, can implement software measurement techniques that can be used to assess the software's reliability during developmental testing, i.e., before the software is delivered. The software manager would do this reliability evaluation by establishing a measurement framework (plan) using the failure data collected by the developer during the product's design phase.

In addition to collecting failure data, other metrics can be collected during the software design phase to provide the evaluator with an early indication of software quality. However, the applicability of these metrics will need to be determined through various metric evaluation techniques. This evaluation will indicate whether a relationship exists between the metric and the quality of the software under evaluation. Examples of these metrics include the number of executable statements, comments (non-executable code), paths, cycles, and total lines of code (total non-commented lines of code). A complete discussion of metric evaluation is beyond the scope of this thesis.

3. Collecting the Data

Without data, the model would be useless and reliability predictions would not be able to be made. For this data collection, a Data Base Management System (DBMS) may prove to be helpful (see Table 1 for the required data elements.) For computational purposes, the file management system of certain software reliability tools (e.g., SMERFS and Statgraphics, which are discussed in Appendix A) are usually adequate. However, to manipulate large amounts of failure and metrics data, a specially designed DBMS may be beneficial. This DBMS engine would allow for data sorting for various analyses and reporting purposes. This is accomplished by identifying the key fields of the data (date, time of failure, type of failure, degree of failure) and relating those fields with others. By using the DBMS's query capability, various statistics and reports can be produced by the touch of a few keys. This data can then be properly formatted to be input into the model and further evaluated for trends.

System ID	**Days # (since start of**	**Problem Report ID**	**Problem Severity**	**Failure Date**	**Module with Fault**	**Description of Problem**

Table 1: Sample Data Collection Format

For each system, there should be a brief description of its purpose and functions. The *Days # field* could also be noted in hours or minutes, as appropriate. It is recommended that the *Problem Report ID* field be coded to indicate Software (S) failure, Hardware (H) failure, or People (P) failure.

A more detailed discrepancy report is found in Appendix A. This detailed report could be implemented by the organization as it becomes more familiar with the Software Reliability Process.

4. Establishing Problem Severity Levels

The organization will need to establish some consistency in describing the faults it discovers. This will allow better analysis and classification of failures in the reliability predictions. Some AIAA (1993) recommended severity level descriptions are as follows:

Level 1. Loss of life, loss of mission, abort mission
Level 2. Degradation in performance
Level 3. Operator annoyance
Level 4. System ok, but documentation in error
Level 5. Error in classifying a problem (i.e., no problem existed in the first place.)

These levels should be recorded as part of Table 1.

Note: Not all defects result in failures.

5. Estimating Model Parameters

Once a model has been chosen to be applicable to a particular system, the necessary model parameters must be estimated using SMERFS (see page 16, Step 11 for a brief discussion of this software tool). For the purposes of this thesis and project, the Schneidewind Software Reliability Model is used. Three parameters are used in this model and will be used for MCTSSA: α, which is the failure rate at the beginning of the testing interval "s," and β, which is the failure rate per failure, and "s," the first interval used in parameter estimation. (Schneidewind, 1995)

6. Selecting the Optimal Set of Failure Data

This stage selects the subset of failure data, starting with the beginning interval, "s" through "t," the last observed interval, that will give the best parameter estimates and the most accurate predictions. It relies on the observation that the software process and product change over time. Therefore, old data may no longer be representative of the current and future state of the process and product and may not be as applicable for reliability predictions as the more recent data. A comprehensive discussion of this factor is provided in Appendix A.

7. Identifying the Operational Profile

The operational profile describes the system's environment. It is usually discussed in terms of modes (single node or multi-node operation), frequency of use of a particular station with each station performing a different function (e.g., Workstation 1 performing database functions, Workstation 2 performing wordprocessing functions), and the frequency of function execution (the amount of time the application has been running). It includes the input variables (e.g., a listing of available equipment or a ship's destination), the functional environment of the program (i.e., a specific function the system is to perform such as sorting the available equipment by minor property number), and the output variable (e.g., a printout of the ship's destinations for the next two months). In this framework, a failure can be seen as a departure of the output variable from what it is expected to be. (Musa, 1987) Of note for this project, a single node configuration is assumed. The applicability of the Schneidewind Software Reliability Model to multi-node configuration is discussed Chapter III of this thesis.

As part of the operational profile, the organization would be using the obtained failure data and calculating the various parameter inputs to be used in the reliability model. A detailed discussion of these parameters is beyond the scope of this thesis; however, the importance and significance of these calculations is further discussed in Appendix A.

8. Making Reliability Predictions

This step is the key to predicting the reliability of the software under evaluation. Each of the listed predictions and the applicability to a managerial decision is described in detail in Section 2 of Appendix A. For completeness, however, the possible predictions resulting from the model application are:

- Time to Next Failure
- Cumulative Failures for a Specified Time
- Remaining Failures and Fraction of Remaining Failures
- Total Failures over the Life of the Software
- Test Time to Achieve Specified Remaining Failures
- Operational Quality

9. Validating the Model

This step evaluates the model to determine if it actually measures what the model is designed to measure. The predicted values are compared to the actual values to make a determination of the model's validity. As an example, if the model predicts that the time to next failure will be two periods, this predicted time would be compared to the actual time. Validation is achieved after certain number and types of predictions have been made with a specified accuracy (e.g., average relative error of $\leq 20\%$).

If, however, the values do not compare favorably, the data used in the model should be carefully examined to identify if anything unusual can be found. If the data appears valid, and the model prediction does **not** match reality, different models would need to be investigated. For the purposes of this thesis and project, the Schneidewind Software Reliability Model will be used exclusively.

10. Making Reliability Decisions

The purpose of implementing a reliability program is to provide the manager with additional information through which he can make informed decisions. Reliability decisions such as "Is the software safe enough to use such that it will not cause or result in loss of life?" can be made as a result of the model's predictions. This particular application can be used in the *Shuttle* software. Here, as an example, the manager must decide whether or not to launch the space shuttle based on the software reliability predictions. For this example, the predicted remaining failures must be less than a specified critical value and the predicted time to next failure must be at least as large at the mission execution time plus some safety margin. [This example will be addressed later in Appendix A using specific numerical examples. It is presented here to provide continuity of thought for the steps in implementing a software reliability program.] For any organization, the predicted software reliability can be key to the managerial decision to accept final delivery of the product or not. If the software is *predicted* to perform within specifications, the software can be accepted by the organization as fulfilling the contractual obligations. If it is predicted to fall short of the desired goals, further discussion may be needed in addition to further testing and evaluation.

11. Using Software Reliability Tools

There are software reliability tools available to make the model calculations easier to achieve. The *Statistical Modeling and Estimation of Reliability Functions for Software*, SMERFS, is a software package available for this purpose. (Farr, 1993) Additionally, Statgraphics, a statistical analysis program, is used to augment SMERFS calculations. Sample SMERFS and Statgraphics sessions are outlined in the Testing Procedures section of Appendix A.

E. SUMMARY OF SOFTWARE RELIABILITY IMPLEMENTATION PLAN

In summary, the first phase in the software reliability engineering (SRE) process is to state the organization's reliability goals. These goals can be ideal or conceptual but must have some basis in reality. A goal of "0%" defects might be the ideal objective, but it would not occur in the real world. Imagining for the moment that it could happen, it would cost an extraordinarily large sum of money to obtain. (Recall Figure 1, the Software Reliability Tradeoff Curve).

The second phase of the SRE involves testing. It is here that the failure data is collected and formatted for inclusion in the model of choice. **It is the data that allows the predictions for reliability to be made.** The test plan used must be consistent with the goals established. If a goal is to have a maximum number of remaining failures set at less than one, then the test plan must be able to predict the remaining number of failures in the software. The tests provide insight into the future -- what may occur as a result of using this software. This insight is used to either forge ahead with actual implementation of the software or return to the drawing board and reassess the system. It will provide an indication as to whether or not additional testing is needed because the results to date may be inconclusive or show an undesirable trend. The test results also allow the manager to prioritize his assets. It can help him to decide where he should assign his resources. Is Module C predicted to be more reliable than Module B? If this is true, he may decide to allocate the majority of his resources to Module B to improve its reliability.

These SRE steps provide the reader with the general overview of the reliability methology that should be carried out as part of the software reliability engineering process. Chapter III provides amplifying information regarding the specific data that must be collected, how it is analyzed by the model, and how the results of the model can be interpreted. It further demonstrates the applicability of the SRE to MCTSSA software systems.

F. COMMERCIAL APPLICATIONS OF THE SRE PROCESS

SRE is not a new phenomenon overtaking the software community. It is a process that is, however, gaining visibility. Several well known organizations currently employ the process as described above. Key among these organizations is AT&T which is considered to be in the forefront of this technology since the early 1970's when John Musa began writing about this topic. The Navy has applied this technology to the capital Trident missile series. NASA (IBM Federal Services Company, now Loral Space Information Systems) has been reporting its reliability results since the early 1980's and has completed several studies on their Space Shuttle system. (Stark, 1992) It is on this latter organization that the majority of examples in Appendices A and B are based. The Schneidewind Software Reliability Model is used extensively by NASA to predict the reliability of the shuttle's onboard system software (Scheidewind, 1992).

III. AN SRE PROCESS CASE STUDY

A. INTRODUCTION

With software reliability frequently seen as an indication of software quality, most organizations are focusing their attention on this difficult to measure concept. The Marine Corps Tactical System Support Activity (MCTSSA), Camp Pendleton, CA is one such DOD organization. Part of this command's mission is to provide "cradle to grave" software support for selected software systems. The key objective of this mission assignment was to establish "a unity of effort in the technical management of software engineering, software design, development and integration and post deployment software maintenance." (MCTSSA, 1994)

B. MCTSSA'S REQUIREMENTS FOR ESTABLISHING AN SRE PROCESS

MCTSSA uses reliability as a measure of an Automated Information System's (AIS's) operational suitability. They define AISs as "multi-functional, distributed systems where functionality is distributed in various servers/workstations around a network." (MCTSSA, 1995) Some of the systems with which they work can provide alternate communication paths and multi-source inputs into servers or workstations, thereby reducing the impact of single point failures. The challenge MCTSSA faces is that the reliability criteria used in the past to measure the suitability of an AIS's field deployability may not be appropriate for the distributed systems being developed today. (MCTSSA, 1995)

The command required a software model that could be used to measure the software reliability of certain AISs during follow-on testing and evaluation. Key in this requirement was determination of the applicability of the selected model to multi-node systems since most reliability models available today apply to single node configurations. As part of this research project, a Software Reliability Training Plan and accompanying Handbook would be provided to the organization. These deliverables (included in this thesis as Appendices A and B) would be used by MCTSSA to serve as references for implementing standard software reliability practices at the command and applying the selected Software Reliability Model.

C. APPLICABILITY OF THE SRE PROCESS TO MCTSSA

The generic steps of the SRE Process, as recommended by AIAA (1993), were discussed in Chapter II of this thesis. These steps can be adopted by any organization which desires to implement a management process to measure the reliability of the software it is evaluating. Through the use of a Software Reliability Model, the command can use the model's results to **predict** the future reliability of the software. This will enable management to get an indication of the delivered product's quality and permit the manager to assign his resources appropriately.

As an organization designed to provide "cradle to grave" software support for selected software systems, MCTSSA can gain invaluable predictions. The results of the model can be used by the command, prior to its acceptance of the finished product from the developer, to make some assessments on the suitability of the software's reliability. Does the reliability meet the requirements stated in the contract? How does the software's **predicted** (future) reliability stand? Is it within accepted tolerances or is it out of range? Will *additional* testing of specific modules be required? All of these questions can be addressed through adoption of the SRE Process and proper selection of a Software Reliability Model.

It was the focus of this thesis to address these issues and provide a recommended strategy for MCTSSA in its reliability assessments using a current software project under development for MCTSSA, specifically, the Marine Air-Ground Task Force II/Logistics Automated Information Systems (or LOGAIS, for short). This system is "a family of coordinated, mutually supporting, automated systems designed to support deliberate and crisis action/time-sensitive planning, deployment, employment, and redeployment of a MAGTF in independent, joint, and/or combined operations." (MAGTF II/LOGAIS, 1992) It is a combination of microcomputer-based systems designed to provide all information necessary for seamless integration with the Joint Operational Planning and Execution System (JOPES). (MAGTF II/LOGAIS, 1995) The functions and specifics of the system are beyond the scope of this thesis. However, since it operates in a microcomputer-based environment it is perfectly suited for application of the SRE process.

The model selected for this project was the Schneidewind Software Reliability Model, one of four models recommended by the AIAA (1993). A discussion of the recommended strategy and the numerical calculations required by the model are included as part of the Handbook provided to MCTSSA (Appendix A) and will not be repeated here. This model is traditionally applied to single node configurations, as was done in this project. However, Section H of this chapter proposes the application of the Schneidewind Software Reliability Model to a multi-node configuration.

D. DATA COLLECTION REQUIREMENTS AND CHALLENGES

As previously mentioned, data collection is the most important and challenging aspect of implementing an SRE process for any organization. Understanding that MCTSSA is not the developer of the software product, it can, however, encourage and/or require the contractor to implement specified practices. This will ensure that the organization obtains the reliability data in the format it desires, during development and prior to formal testing and acceptance. This will enable MCTSSA to apply the statistical processes of the Schneidewind Software Reliability Model to get an approximation of the software's *predicted* reliability.

There are eight steps identified by the AIAA (1993) as a means for effective data collection:

- Establish the Objectives
- Plan the Data Collection Process
- Acquire Data Collection Tools
- Provide Training
- Perform Trial Run
- Implement the Plan
- Monitor the Data Collection and Use the Data
- Provide Feedback to all Parties

Most of these steps are beyond the control of the independent software tester, MCTSSA, but can be discussed during contract negotiations. MCTSSA can require the contractor to provide fault data collected during the software design, which can then be used by MCTSSA to ascertain the software's reliability status. Of note in this area is the fact that data collection costs money. The organization desiring the test data should be specific about the type of data it desires as part of their SRE implementation strategy. Since there may be a tendency to want to collect everything, the decision makers must tradeoff cost of collection (including the burden on data collectors) with return (Stark, 1992).

In the development of this research project, MCTSSA obtained a database of compiled defect data from the contractor. This database, **Software Edge's** *Defect Control*

Systems for Windows, Version 2.10, contained all the defect data the developer recorded on the software during its design. As a database, the software provided a query capability which was used extensively to draw out the appropriate data for inclusion in the Schneidewind Software Reliability model. This data was the **number of defects** recorded during an interval, one day, by date the defect was submitted to the database. This data was then formatted chronologically (by a workday, not calendar day, since defects were only listed during normal working hours) for inclusion in a table for easy of readability and analysis. This date sequencing permits reliability predictions to be made. All of the 4584 defects from November 11, 1994 through May 17, 1995 are listed in Appendix C. A determination was made to group the data by five-day increments (to simulate the typical work week).

In addition to the data not being recorded in the database by true failure date, there are other challenges the LOGAIS database presented: (1) the data was not true failure data because it was not recorded in execution time when failures occur; rather it was recorded by administrative convenience, by batches at the end of the workday; and (2) the data shows large swings in daily defect count (Figure 2) not showing the expected decrease in number of faults as time progresses (recall Figure 1). (Schneidewind, 1995b)

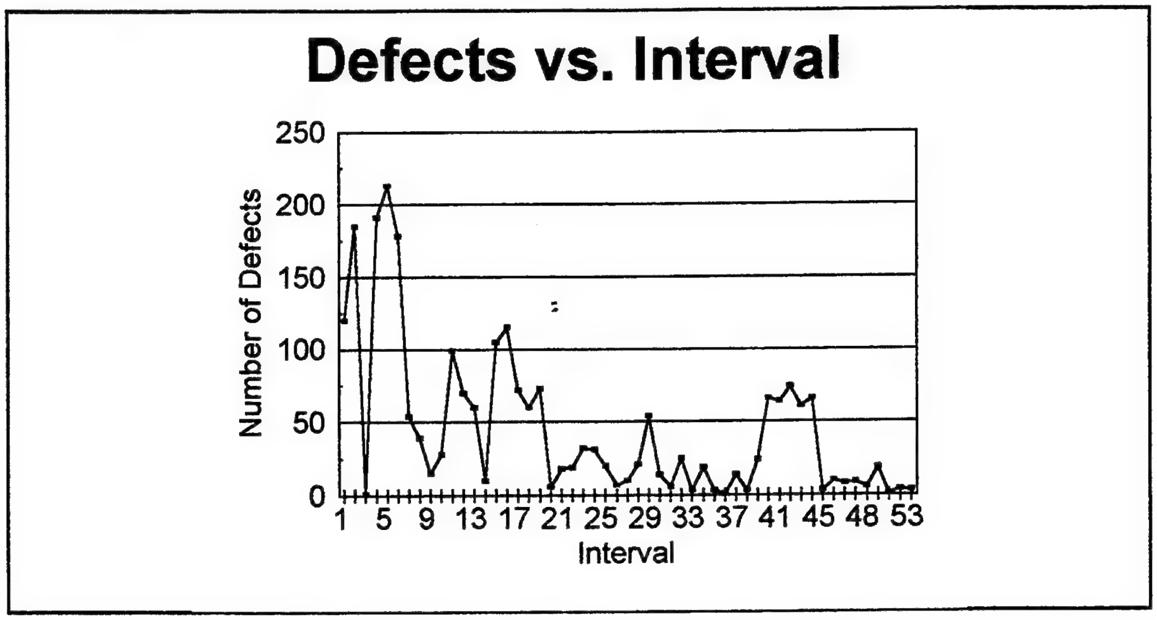


Figure 2. Defect Count vs. Time Interval

If, however, the data is averaged over five day intervals, a decreasing trend does emerge but there are still some unanticipated peaks and valleys. This trend can be seen in Figure 3.

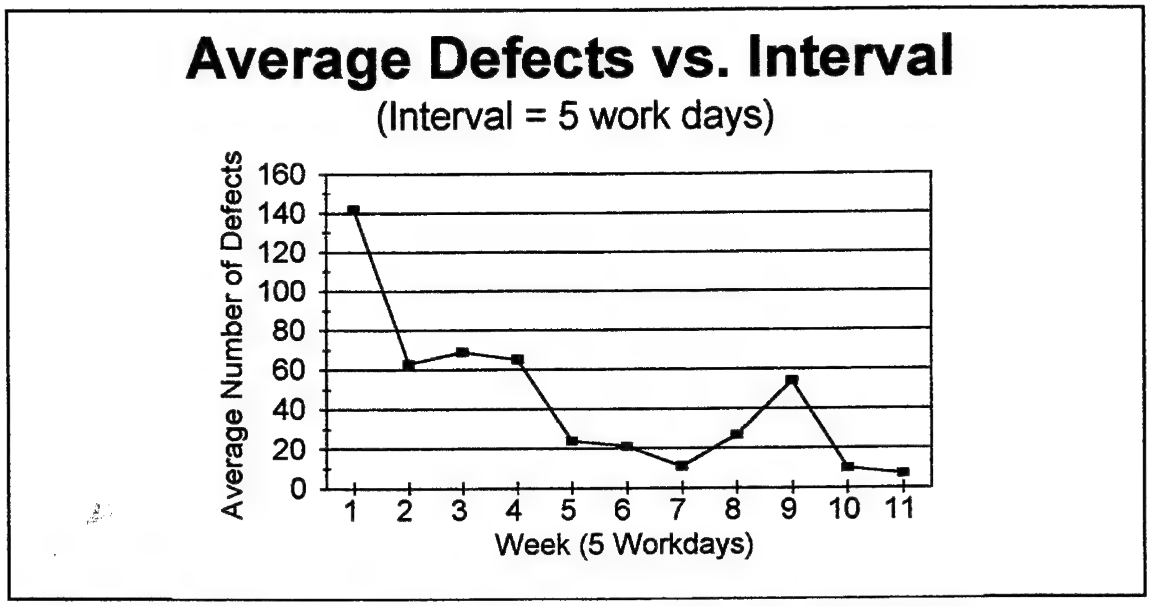


Figure 3. Averaged Defects vs. Typical Five Day Work Week

If the data is smoothed out even further by using a “moving average” of 30 days, a decreasing trend is seen. This is shown in Figure 4.

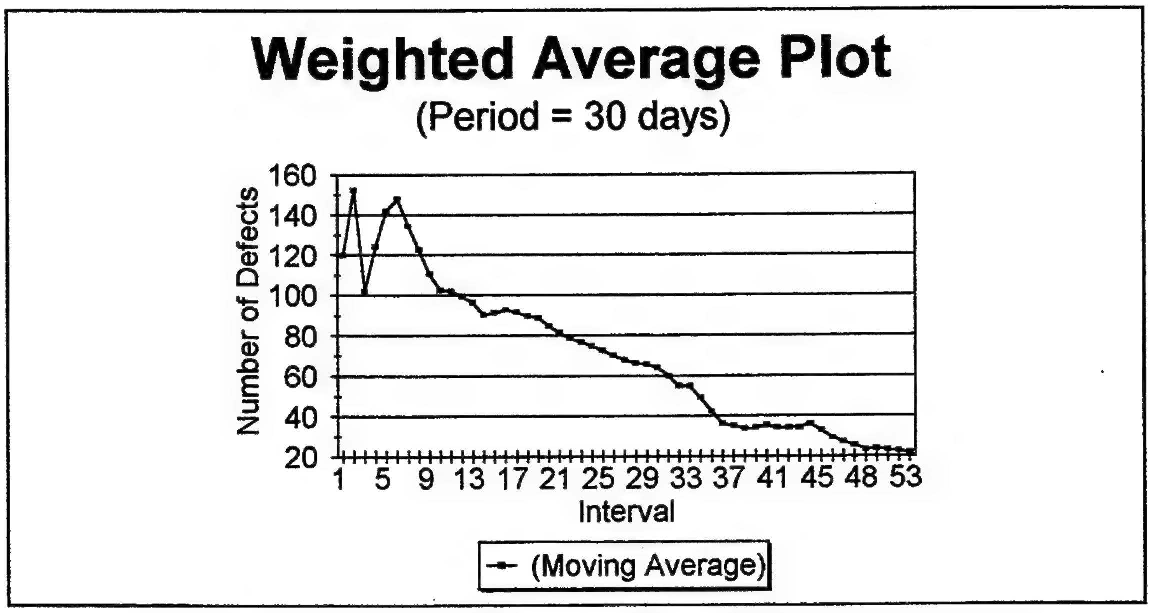


Figure 4. 30 Day Weighted Average of Number of Defects Recorded

These views of the data show the plausibility of using smoothed data as the input to the reliability model instead of using raw data obtained directly from the LOGAIS database. In this test of the data, raw LOGAIS data (the normal approach) did produce fair predictions. Further studies using smoothed data versus raw data would need to be completed and compared to demonstrate if any trends or accuracies are affected by the choice of data used.

This section will discuss the actual application of the Schneidewind Software Reliability Model and the inputs required to obtain meaningful results. **A comprehensive discussion of the model parameters, inputs, and results can be found in the *MCTSSA Software Reliability Handbook*, included as Appendix A in this thesis.**

E. MODEL APPLICATION AND ITS RESULTS

In order to obtain the most accurate model parameters, both one-day and five-day intervals were used. By comparing the Mean Square Errors (MSE) of these two intervals, it was seen to favor the five-day cycle. Also varied were the length of the input data recorded in the range t = 20, 55 for one-day intervals and in the range t = 13,20 for five-day intervals, and used the value of the MSE to determine the optimal value for "t." (Schneidewind, 1995b)

1. Defect Count Predictions

As previously mentioned, it is advantageous for management to know what the predicted reliability of the software is to help estimate additional testing time needed. This also allows for proper resource management, i.e., assignment or not of a greater number of personnel. This prediction can be achieved through the model's outputs for the predicted number of defects in a selected interval range. This interval range can vary, but is seen as an interval of time in the future, that is, "how many defects are predicted to occur in the next two work weeks?" This was accomplished through the application of Equation 1 in SMERFS:

$$\mathbf{F(t_1,t_2)=(\alpha/\beta)[1-exp(-\beta(t_2-s+1))]-X_{s,t1}} \qquad \mathbf{(1)}$$

"where (1) is the predicted number of defects in the interval range t_1,t_2; s is the optimal interval to start using defects for the estimation of α and β; and $X_{s,t1}$ is the observed number of defects in the range s,t_1." Here t_1 is defined as t, the end of the parameter estimation range, and t_2 is the prediction interval. The results obtained showed that t = 30 gave relatively good predictions as can be seen in Figure 5. (Schneidewind, 1995b)

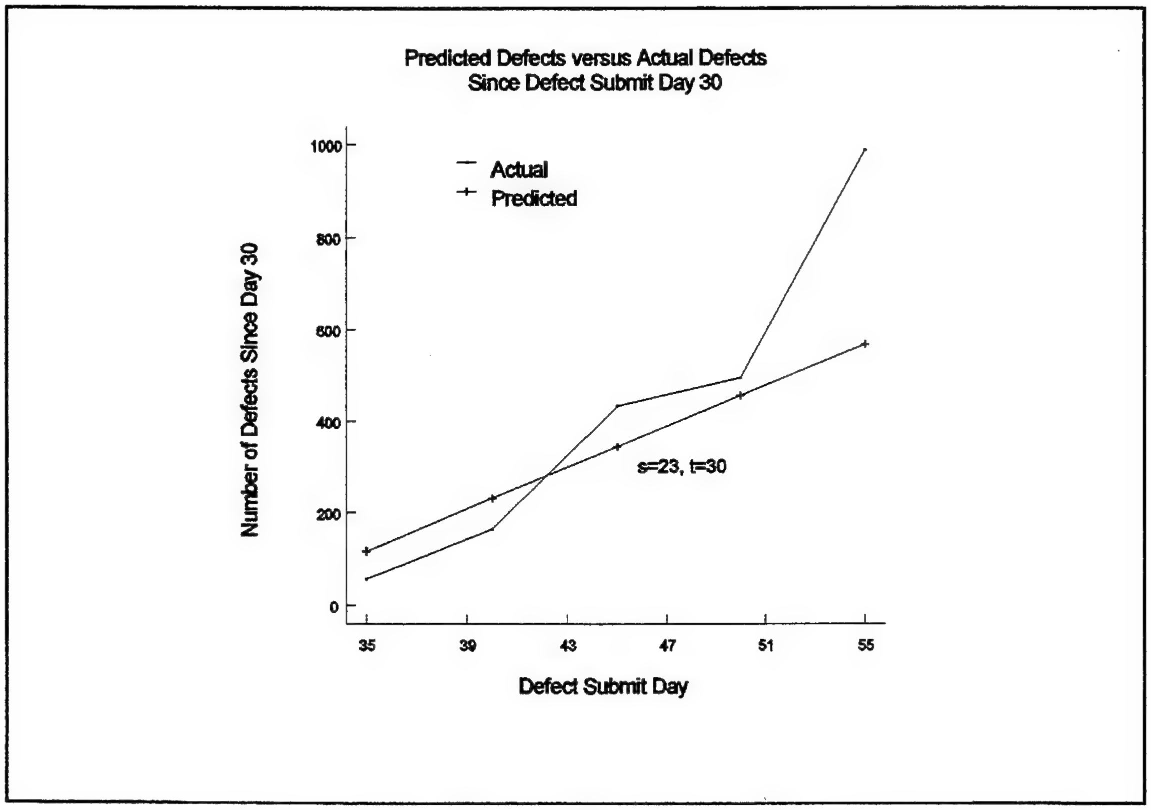


Figure 5. Predicted Defects vs Actual Defects

To determine "s," Equation 2 was used (via SMERFS):

$$\mathbf{MSE}_{F} = \frac{\sum_{i=s}^{t} [\alpha/\beta(1-\exp(-\beta(i-s+1)))-X_{s,i}]^2}{t-s+1} \qquad \textbf{(2)}$$

This equation calculates the MSE for defect counts and cumulative defect counts. "It computes the sum of the squared differences between model predictions and actual cumulative defect counts $X_{s,i}$ in the range $s \le i \le t$." Here, s = 23 was optimal for t = 30. (Schneidewind, 1995b)

Figure 5 illustrates the prediction of (1), starting at t = 30 and predicting for $t_2 = 35$, 40, 45, 50, and 55 days, where these represent predicted defects in the intervals 5, 10, 15, 20, and 25 days, respectively, from t = 30. It is seen that the predictions appear good until $t_2 = 55$ when the actual defect count takes a sharp turn upward. "This is **counter** to what one would expect -- a *decrease* in the *rate* of finding defects as testing continues because the defects become harder to find. When this occurs, it indicates the need for using more of the available data, re-estimating the parameters, and repeating the predictions." (Schneidewind, 1995b)

2. Cumulative Defect Count Predictions

As part of a proactive management practice in software reliability, it is advantageous for the manager to be aware of the cumulative count of defects predicted in the software. This information can be used similarly to the defect count predictions but gives a better indicator of the degree of testing problems encountered to date. For this calculation, Equation 3 was used through Statgraphics:

$$F(T)=(\alpha/\beta)[1-\exp(-\beta(T-s+1))]+X_{s-1} \tag{3}$$

where X_{s-1} is the defect count in the range 1,s-1.

The results of this calculation did not produce a single curve that accurately matches the true count of cumulative failures. However, upper and lower bound curves were generated as seen in Figure 6.

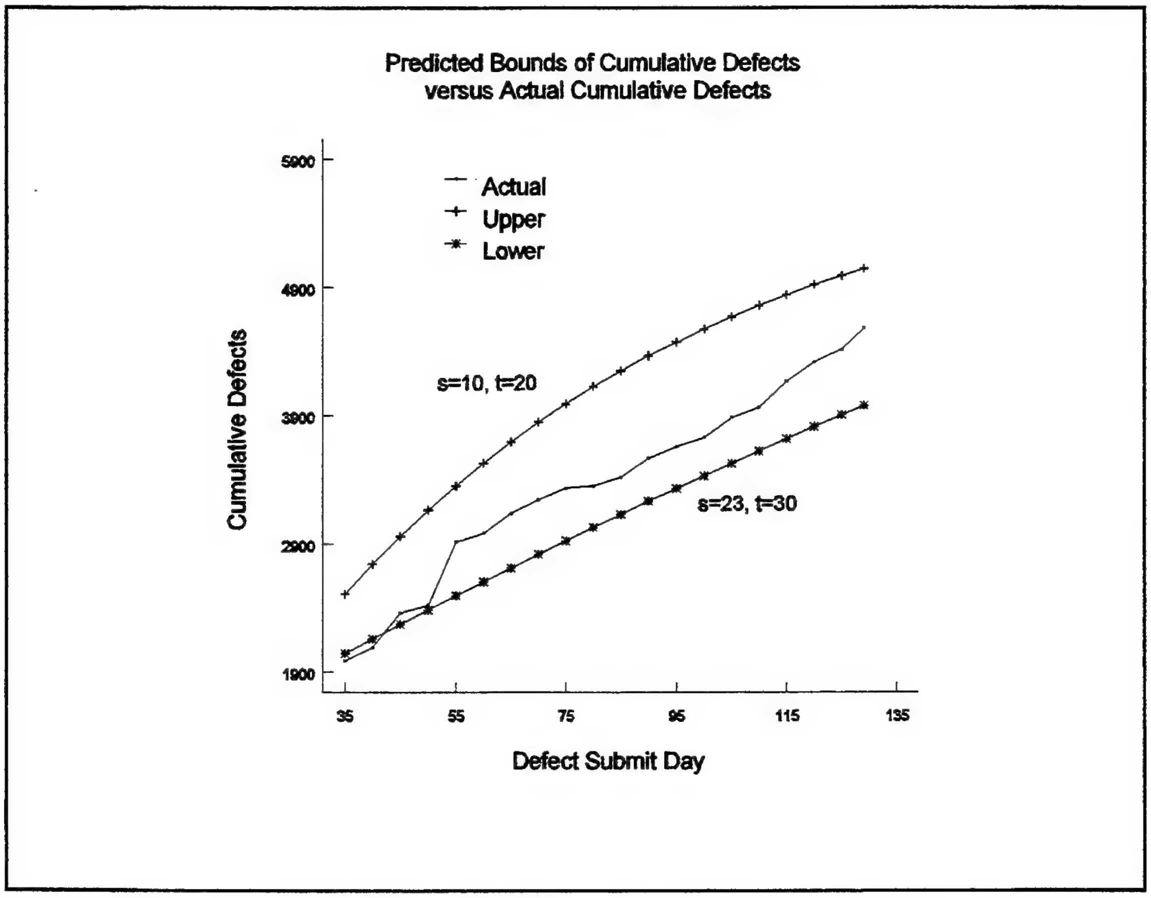


Figure 6. Predicted Bounds of Cumulative Defects vs. Actual Defects

"The concept of bounding is important in prediction because the manager would like to know within what limits a quantity is likely to fall." Figure 6 shows that the predicted cumulative defect count for day 129 (May 17, 1995) would fall between 3978 and 5047. The true cumulative defect count for that date is 4584. (Schneidewind, 1995b)

3. The Amount of Time Needed to Find the Defects

Each of the previous calculations focuses on the issue that given a specific time interval - for example, the next two work weeks - how many defects would we predict to be discovered? A corollary to this question can be proposed. "How much time would it take to find a specific number of defects?" Equation 4, implemented in SMERFS can be used to help answer this question.

$$\mathbf{T_F(t)=[(\log[\alpha/(\alpha-\beta(X_{s,t}+F_t)])/\beta]-(t-s+1)} \\ \mathbf{for\ (\alpha/\beta)>(X_{s,t}+F_t)} \qquad \mathbf{(4)}$$

where (4) is the predicted time (intervals) until the next F_t defects are found, t is the current interval, and $X_{s,t}$ is the cumulative number of defects observed in the range s,t. s = 23 and t=30 were used to produce Figure 7. (The rationale for selecting the proper t and s can be found in Appendix A.) Since the defect count is cumulative, F_t, the time to find the defects increases with time, as can be seen in Figure 7. This is expected since it will take longer to find defects as time passes. The predicted and actual values are comparable until Day 55, when there is an upward increase in predicted time to find the defects. As before when this occurred, there is a need to use more available data, re-estimate the parameters, and repeat the predictions. (Schneidewind, 1995b)

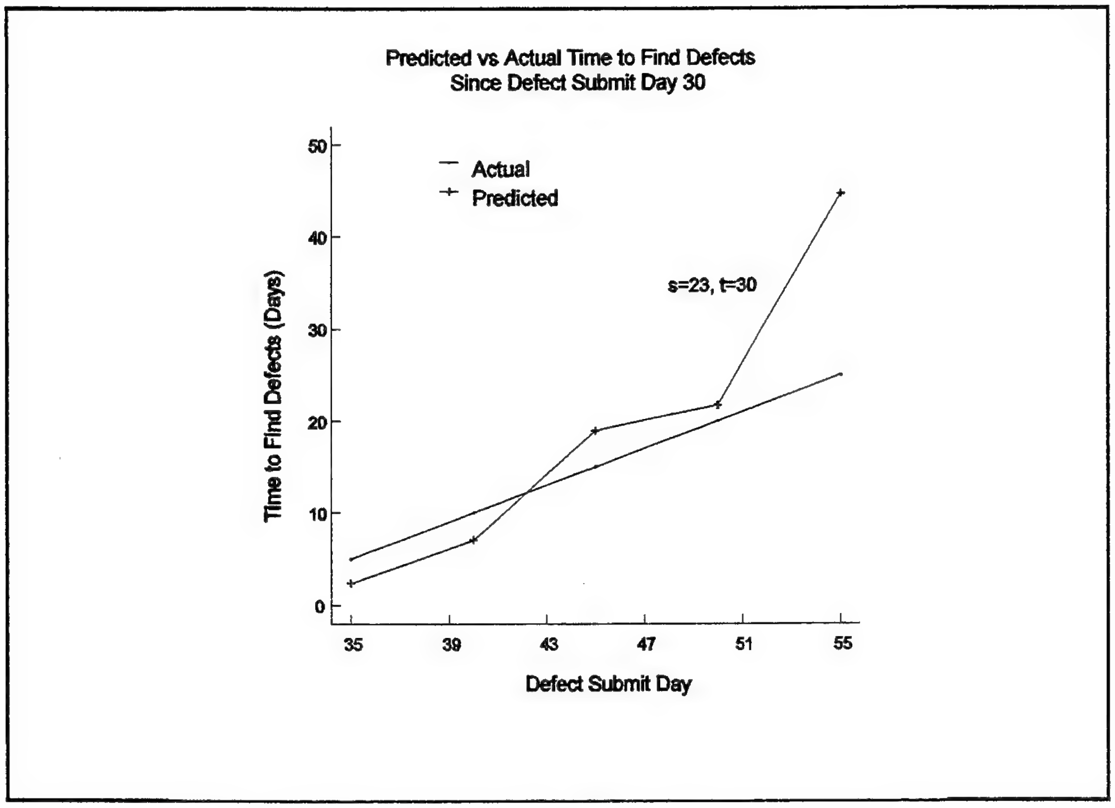


Figure 7. Predicted vs. Actual Time to Find Defects

4. Results

Based on the above predictions, it appears feasible to employ a software reliability model to data obtained for MCTSSA's LOGAIS project. The Schneidewind Software Reliability Model gave predictions comparable to actual defect counts. However, in future calculations, smoothed data would need to be employed to obtain better prediction accuracy.

F. USE OF THE SRE PROCESS IN A MULTI-NODE CONFIGURATION

This section will present a **proposed** model for use by MCTSSA with its **system** survivability predictions (multi-node configurations). This is in contrast to the previously discussed single node survivability concepts presented in the previous sections of this chapter. The two models take into account different possible system configurations and the impact of server and client failures. These configuration setups can be found in Figures 8 through 10. Of note, this section does not present any actual calculations using MCTSSA test data. It only presents concepts that need to be further evaluated and tested. However, this section can help explain some of the uses for the Schneidewind Software Reliability Model, its relevance to the multi-node configuration concept, and its applicability to an organization's system design and configuration.

1. Model 1

In this situation, *there are critical clients: clients with critical functions* (e.g., network communication) that must be kept operational for the system to survive. There are also non-critical clients with non-critical functions (e.g., data base query). These clients also act as a backup for the critical clients. The system does not fail unless (1) all the non-critical clients fail **and** one or more critical clients fail, **or** (2) one or more servers fail. The model concepts are illustrated in Figures 8, 9, and 10 where there are two servers, five critical clients (C1...C5), and five non-critical clients (C6...C10). In Figure 8 one non-critical client, C6, fails; therefore the system survives. In Figure 9 one of the servers, S1, fails; therefore the system fails. Lastly, in Figure 10 one of the critical clients, C5, fails and all of the non-critical clients, C6 ... C10, fail; therefore the system fails.

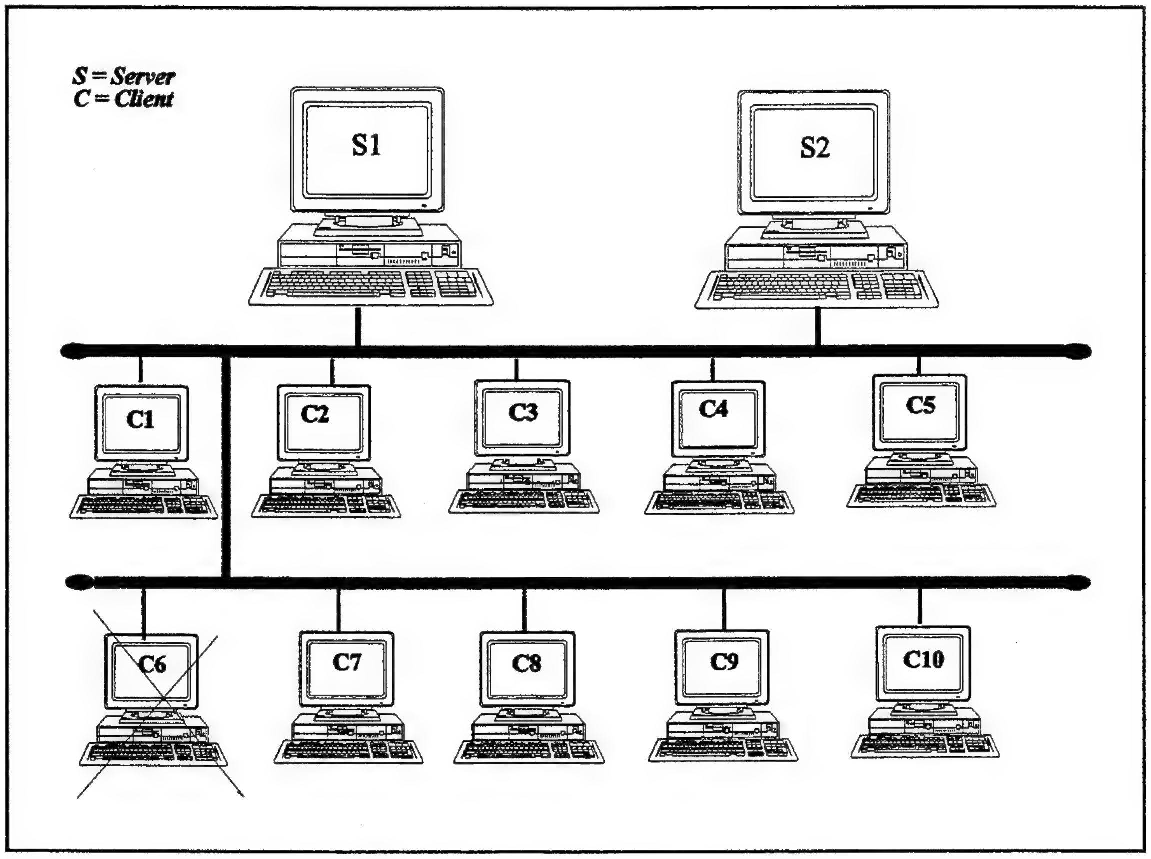


Figure 8. Surviving Configuration

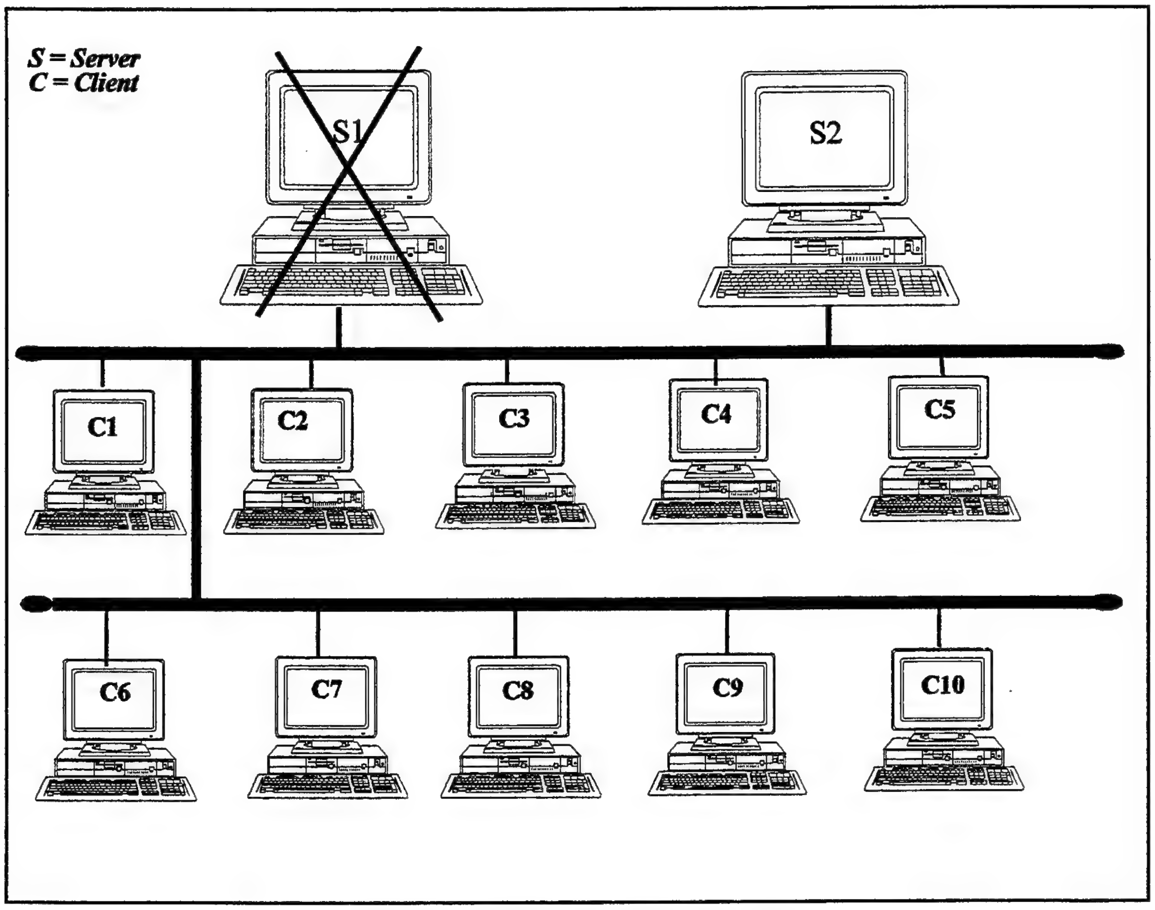


Figure 9. Failing Configuration # 1

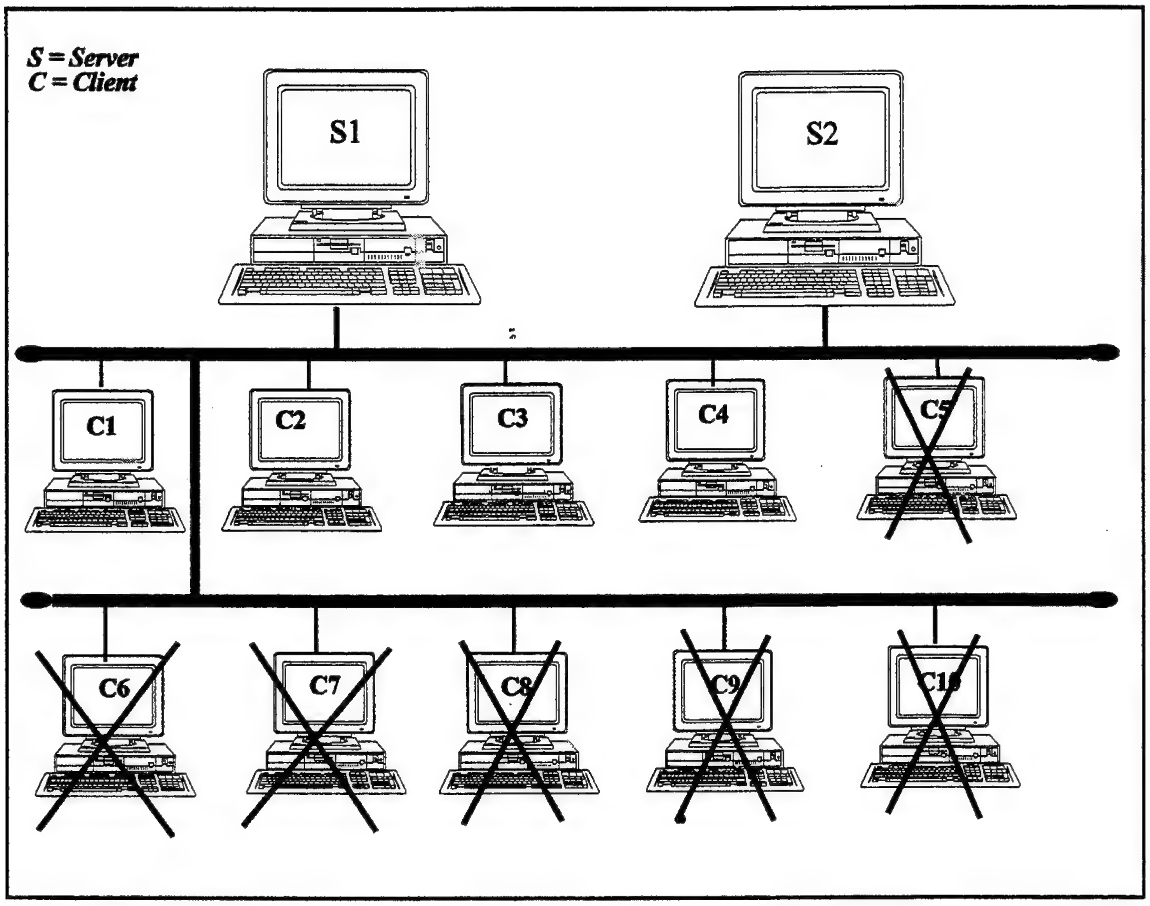


Figure 10. Failing Configuration # 2

Schneidewind (1995b) proposes the following descriptions to help explain the calculations and concepts involved in this model:

a. **Client or server failure**: the application software or operating system in the node ceases to function and the client or server is lost to the distributed system, as a result of a software failure.

b. **System failure**: the system ceases to be operational because either: (1) all non-critical clients fail *AND* one or more critical clients fail *OR* (2) one or more servers fail.

c. **$N_n(t)$**: The number of non-critical clients available in the system at time t, where $N_n(0)$ is the number of non-critical clients at the start of system operation. If a non-critical client fails, the system can continue to operate -- in a degraded mode -- as long as none of the servers or critical clients fail. In this situation, the function that had been operational on the failed non-critical client can be continued on another client of this type and $N_n(t)$ is decreased by one.

d. **N_c**: The number of critical clients used in the system. If a critical client fails, the system fails, *if there are no non-critical clients available* on which to run the critical client. A change in software configuration may be necessary on the former non-critical client in order to run the critical client. The former non-critical client becomes a critical client, $N_n(t)$ is decreased by one, and the system is run in a degraded mode. As long as the system *remains operational*, N_c is constant.

e. **N_s:** The number of servers used in the system. If a server fails, the system fails.

f. The probability that **all** $N_n(t)$ have failed by time t, given that the software fails, is (Schneidewind, 1995b):

$$\mathbf{P_n(t)=(p_c)^{Nn(t)}}, \qquad \textbf{(5)}$$

where p_c is the probability that the software failure causes a client failure: p_c=probability (client fails | software fails). Equation (5) assumes that client failures are independent. This is the case because a failure in one client's software would not cause a failure in another client's software. However it is possible that a failure in server software could cause a failure in client software, such as a client accessing a server that has corrupted data. The extent that this could happen depends significantly on whether the client software has been designed to protect against such occurrences. Unless information can be obtained about such occurrences, this factor will be ignored. The probability p_c can be estimated empirically as the ratio of: (client down time caused by software failure)/(scheduled client operating time).

g. The probability that **one or more** N_c fail, given that the software fails, is (Schneidewind, 1995b):

$$\mathbf{P_c=1-(1-p_c)^{Nc},} \qquad \mathbf{(6)}$$

h. The probability that **one or more** N_s fail, given that the software fails, is (Schneidewind, 1995b):

$$\mathbf{P_s=1-(1-p_s)^{Ns},} \qquad \mathbf{(7)}$$

where p_s is the probability that the software failure causes a server failure: p_s=probability (server fails |software fails). Equation (7) assumes that server failures are independent. This is the case because a failure in one server's software would not cause a failure in another server's software. However it is possible that a failure in client software could cause a failure in server software, such as a client with corrupted data accessing a server. The extent that this could happen depends significantly on whether the server software has been designed to protect against such occurrences. Unless information can be obtained about such occurrences, this factor will be ignored. The probability p_s can be estimated empirically as the ratio of: (server down time caused by software failure)/(scheduled server operating time).

i. Combining (5), (6), and (7), the probability of a system failure by time t, given that the software fails, is (Schneidewind, 1995b):

$$\mathbf{P_{sys}(t)=(P_n(t))(P_c)+P_s=[[(p_c)^{Nn(t)}][1-(1-p_c)^{Nc}]]+[1-(1-p_s)^{Ns}]} \qquad \mathbf{(8)}$$

2. Model 2

In this situation, *there are only non-critical clients $N_n(t)$.* However there is a minimum number N_{nm} of these clients that must remain operational for the system to survive. Therefore the number that could fail, N_f, and cause a system failure is $N_f \geq N_n(t)-(N_{nm}-1)$. Thus if there were $N_n(t)=10$ clients at time t, and $N_{nm}=3$ clients minimum to keep the system operational, a failure of eight or more clients would reduce the number of clients to less than three.

a. In general the probability of falling below N_{nm} by time t is (Schneidewind, 1995b):

$$\sum_i [N_n(t)!/[(i!)(N_n(t)-i)!]](p_c^i)(1-p_c)^{(Nn(t)-i)} \qquad (9)$$

where $i=N_n(t)-(N_{nm}-1), \ldots, N_n(t)$.

b. Combining (9) and (7), the probability of a system failure by time t, given that the software fails, is (Schneidewind, 1995b):

$$P_{sys}(t)=\sum_i [[N_n(t)!/[(i!)(N_n(t)-i)!]](p_c^i)(1-p_c)^{(Nn(t)-i)}]+[1-(1-p_s)^{Ns}] \qquad (10)$$

IV. CONCLUSIONS

A. PROBLEMS ENCOUNTERED IN THE SRE DEVELOPMENT PROCESS

In addition to the data not being recorded in the developer's database by true failure date, there were other challenges the LOGAIS database presented: (1) the data was not true failure data because it was not recorded in execution time when failures occur; rather it was recorded by administrative convenience, by batches at the end of the workday; and (2) the data showed large swings in daily defect count not showing the expected decrease in number of faults as time progresses. The first problem required manual intervention to sort the defect data chronlogically using the database's query capability. This data then was entered into a table for easy readability. The second problem was overcome by using smoothed data as the input to the reliability model instead of using raw data obtained directly from the LOGAIS database. The smoothed data was obtained by using a moving average over thirty day periods.

B. SUMMARY OF FINDINGS

Based on the single-node reliability predictions discussed in Chapter 3, it appears feasible to employ a software reliability model to data obtained for MCTSSA's LOGAIS project. The Schneidewind Software Reliability Model gave predictions comparable to actual defect counts. However, in future calculations, smoothed data would need to be employed to obtain better prediction accuracy.

Using the multi-node configuration scenario, it appears feasible to develop a system software model for distributed systems. The next step would be to integrate equations (8) and (10) into the *Schneidewind Software Reliability Model*. MCTSSA would then need to collect **system** failure data in addition to defect data to support model validation. Additionally, it would be necessary to know not only that a defect occurs but whether the defect causes a system failure. Information would be needed about typical values for p_c and p_s, and an indication of which applications can be represented by Model 1 and which can be represented by Model 2.

C. BENEFITS ACCRUED FROM THE PROJECT

Establishing a software reliability engineering program will improve the reliability of software delivered to the field through reliability predictions. Additionally, the MCTSSA SRE program demonstrates the use of developmental testing results to predict reliability during the test phase and the need to continuously obtain software failure data for future reliability modeling and predictions.

In today's environment of LAN-based distributed systems, there is a need for a software reliability model that can provide management with insight into the predicted survivability of the system. The adaptation of the Schneidewind Software Reliability Model for use with multi-node configurations provides the organization with such a software evaluation tool.

MCTSSA SOFTWARE RELIABILITY HANDBOOK

FINAL VERSION:
January 10, 1996

Dr. Norman F. Schneidewind
LCDR Judie A. Heineman

Naval Postgraduate School
Code SM/Ss
Monterey, California 93943

Voice: 408-656-2719
Fax: 408-656-3407

Internet: schneidewind@nps.navy.mil

TABLE OF CONTENTS

SECTION 1: IMPLEMENTING AN SRE PROGRAM

A. PURPOSE:

The purpose of this handbook is threefold. Specifically, it:

- Serves as a reference guide for implementing standard software reliability practices at Marine Corps Tactical Systems Support Activity and aids in applying the software reliability model
- Serves as a tool for managing the software reliability program
- Serves as a training aid

B. INTRODUCTION

Representing the "intellectual effort" of its authors, software includes not only the source code, but the supporting documentation and test results. With this in mind, software is a complex concept to evaluate and measure. Trying to predict its reliability is just as challenging.

Reliability is seen as the ability of a system to perform as expected under specific conditions for a specified period of time. This also includes the "**probability** that the software will not cause the failure of a system for a specified time under specified conditions." (AIA93) This concept must be matched with appropriate measurement techniques that provide a mechanism to evaluate the software's ability to perform.

Software Reliability Engineering (SRE) is a new discipline that is maturing as more organizations see the need to develop standard reliability practices. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as "the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems." (AIA93)

C. DEFINITION OF FAULT MEASUREMENT

As with any intellectual product, errors in design may occur. An error can be defined as "a discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition." (AIA93) In software, these errors may appear while completing requirements formulation or, as is often the case, during design, coding, and testing the product. The software development process should include measures to discover and correct faults resulting from these errors. [In this context, faults are defined as "defects in the code that can be the cause of one or more failures." (AIA93)]

These measures can address reviews, audits, screening by language-dependent tools, and several layers of testing. One way to reduce the number and criticality of errors is by modeling the effects of the remaining faults in the delivered product. This can be achieved through a dedicated measurement process by which each defect or fault is noted and formally recorded for inclusion in the reliability model. (AIA93) As a point of clarification, a fault is technically different from a failure. A failure can be defined as "the inability of a system or system component to perform a required function within specified limits" or the "departure of program operation from program requirements." (AIA93) In simpler terms, a fault usually leads to a failure.

D. MANAGERIAL IMPACT OF FAULT MEASUREMENT

Handling, identifying and correcting faults is a significant concern for the manager because the entire software reliability process is expensive. "It also impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time and peripherals requirements)." (AIA93) This addresses the key issue regarding SRE -- *it provides the* *manager with information about which he can make informed decisions.* There will always be a tradeoff between reliability, frequently referred to as the failure rate, and cost. (Cost is directly related to testing time). The manager will need to decide on a certain level of reliability for the product, resulting in a set cost. Thus, higher reliability will result in a higher cost. The converse is also true.

In general, the failure rate of a software system is seen as a curve with a decreasing slope which results from the identification and removal of errors as time passes. It is the primary purpose of reliability modeling to define the shape of this resulting curve using statistical methodologies. The model used in these reliability assessments can provide prediction information regarding the software execution time needed to discover a specified number of faults, or predict the time period when the next fault will occur. Figure 1 provides a sample software reliability curve that can be generated by using a software reliability model. (AIA93)

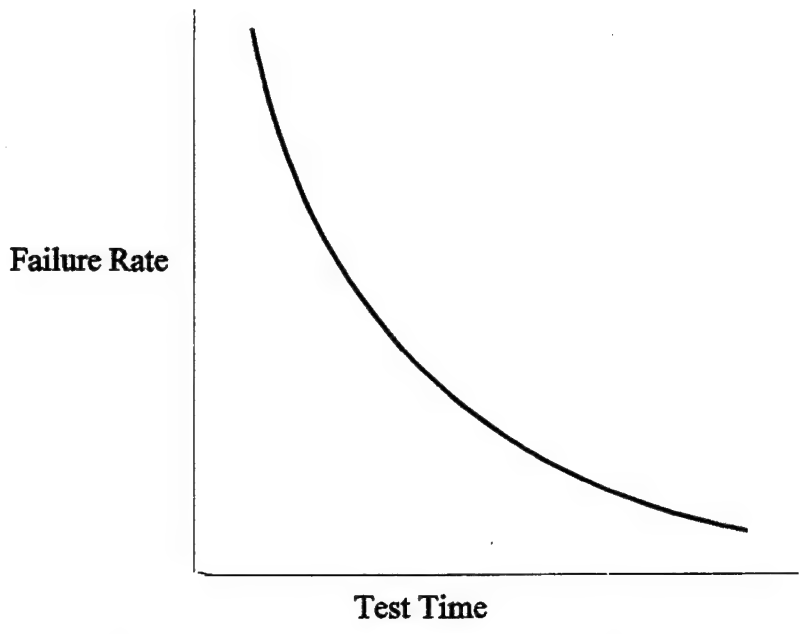


Figure 1: Software Reliability Tradeoff Curve

E. COMPONENTS OF AN SRE PROGRAM

A successful software reliability program does not consist of just a model. It also consists of the support structure: reliability requirements, reliability measurements to meet those requirements, data collection procedures to obtain the necessary data, definition of severity levels of failures, applications of reliability predictions, interpretation of model predictions, and user feedback for model improvements. Although the conceptualization of the model does not occur in a sequence of steps as mentioned above, its *implementation* does. The practitioner can best understand this process from a description of the chronology of implementing and applying the model. Therefore, this approach will be used in explaining the process. To illustrate the process, many equations, figures, and tables will be used. Many real-world - actually *out-of this-world* - examples from the *Space Shuttle* will be used, because the process can be illustrated with real data and real predictions. However, it should not be concluded that the examples are not applicable to MCTSSA; they are. The approach is generic and its feasibility can be tested against MCTSSA systems. The *Shuttle* is a safety critical system where human life and expensive equipment are at risk. This is also the case with MCTSSA systems.

Failure data is preferred to defect data for both empirical reliability assessment and reliability prediction using a model, because the former is a "departure of program operation from program requirements" observed while the program is *executing*, and includes chronologically ordered *test start time* or *operation start time* and *failure occurrence time,* whereas defect data do not contain this time record. Defect data are used more for administrative control to ensure that defects have been resolved than as data for reliability assessment and prediction. However in some systems , such as the Marine Corps' LOGAIS, only defect data are available. In this case the "reliability predictions" will not be as accurate as when failure data are available, but useful predictions can be made nevertheless. Examples of such predictions for LOGAIS are shown in Section 4.

The existing methodology is based on the *Schneidewind Software Reliability Model* (SCH93, SCH75), one of the four models recommended in the *AIAA Recommended Practice for Software Reliability* (AIA93). The validation is based on the fact the model is used to *assist* in assessing the reliability of the *Shuttle* flight software. According to Ted Keller, Manager, Project Coordination, Onboard Shuttle Software Systems, Loral Space Information Systems: "The *Shuttle* software project is experimenting with a promising algorithm which involves the use of the *Schneidewind Software Reliability Model* to compute a parameter: *fraction of remaining failures* as a function of the archived failure history during testing and operation" (KEL95). Obviously remaining failures, fraction of remaining failures and time to next failure would not be used to the exclusion of other approaches in making reliability assessments. These metrics would be combined with process procedures such as inspections, defect prevention, project control boards, process assessment, and fault tracking, to provide a quantitative basis for achieving reliability objectives. (BIL94)

The standard practices described under *Implementing a Software Reliability Program* are essentially those recommended in the *AIAA Recommended Practice for Software Reliability* (AIA93) and the *ANSI/IEEE Standard for a Software Quality Metrics Methodology* (IEE93).

F. IMPLEMENTING A SOFTWARE RELIABILITY PROGRAM

Implementing a software reliability program is a two-phased process. It consists of (1) identifying the reliability goals and (2) testing the software to see how it conforms to the stated objectives. The reliability goals can be ideal or conceptual, e.g., zero defects, but should have some basis in reality. The testing phase is the most complex since it involves the actual collection of raw defect data and molding the data to fit the selected model.

With these phases being the stated objective, the following steps should be considered by the organization as it begins to develop a software reliability program. These steps provide a "cookbook" approach to the SRE process and are ordinarily followed sequentially. Each step will be discussed briefly to provide a general understanding of the purpose of each phase. Stages that require numerical calculations and application of specific model parameters will be noted. Discussion of those parameters will be deferred until Section II of this handbook: The Basic Concepts Used in the Schneidewind Model of the handbook.

The SRE steps are:

- State the Reliability Requirement
- Establish a Measurement Framework
- Collect the Data
- Establish Problem Severity Levels
- Estimate Model Parameters
- Select the Optimal Set of Failure Data
- Identify the Operational Profile
- Make Reliability Predictions
- Validate the Model
- Make Reliability Decisions
- Use Software Reliability Tools

Step 1: State the Reliability Requirement

In this step, the software manager should describe the condition that must be fulfilled for the software to be considered satisfactory (reliable). This is a purely subjective, managerial decision. An example of such a requirement may be the following statement: "The product will have no software failure that would result in loss of life, loss of mission, or cancellation of mission."

Step 2: Establish a Measurement Framework

One approach the organization could employ would be to take the software from the developer at delivery and run it on its own systems and see how well, or poorly, it performed. However, if the manager adopted this approach and waited until the software was delivered to him and then began testing, many months could possibly be wasted if the software is deemed unreliable. In the ideal world, he would have some indications of the system's performance before it was delivered to him. Although this is not an ideal world, the manager does have at his disposal some techniques he can use to get a "feel" for how the software will perform once it is delivered. He would do this by establishing a measurement framework or plan using the fault data collected by the developer during the product's design phase.

The organization should consider a comprehensive measurement plan that would include *indirect* measures of quality like problem report counts, size and complexity metrics. Figure 2 captures this idea. In this diagram, *Level 1* shows the most direct measurement (e.g., a *time between failures*). These are the metrics that can be captured directly by the use of a wall clock and the continuous running of the software. *Level 2* shows an indirect measurement (e.g., *discrepancy report count*) one level removed from the direct measurement. At this level a report is written whenever a discrepancy is observed between the required operation and the actual operation of the software. Most of these reports are derived from static analysis (i.e., inspections), although these reports could record the fact that a failure has occurred; however there would be no data about when tests and operations started and when failures occurred. Hence, it would not be possible to directly calculate the time between failures. Finally, *Level 3* shows an indirect measurement two levels removed from the direct measurement (e.g., *size* and *complexity*). These are the basic attributes of the software itself. How many lines of code were developed? How complicated are the routines in the program? Traditionally, the more complicated the coding, the more likely faults will appear.

The advantage of *Level 1* measurements is that they are the most accurate representations of reliability; their disadvantage is that they cannot be collected until the software is tested. Conversely, the indirect measurements are less accurate as representations of reliability, but they can be collected earlier in the development process. This permits an early indication of the reliability of the software.

In addition to collecting failure data, other metrics can be collected during the software design phase to provide the evaluator with an early indication of software quality. However, the applicability of these metrics will need to be determined through various metric evaluation techniques. This evaluation will indicate whether a relationship exists between the metric and the quality of the software under evaluation. Examples of these metrics include the number of executable statements, comments (non-executable code), paths, cycles, and total lines of code (total non-commented lines of code). A complete discussion of metric evaluation is beyond the scope of this handbook. An example of metrics application can be found in [SCH92a, WAR94].

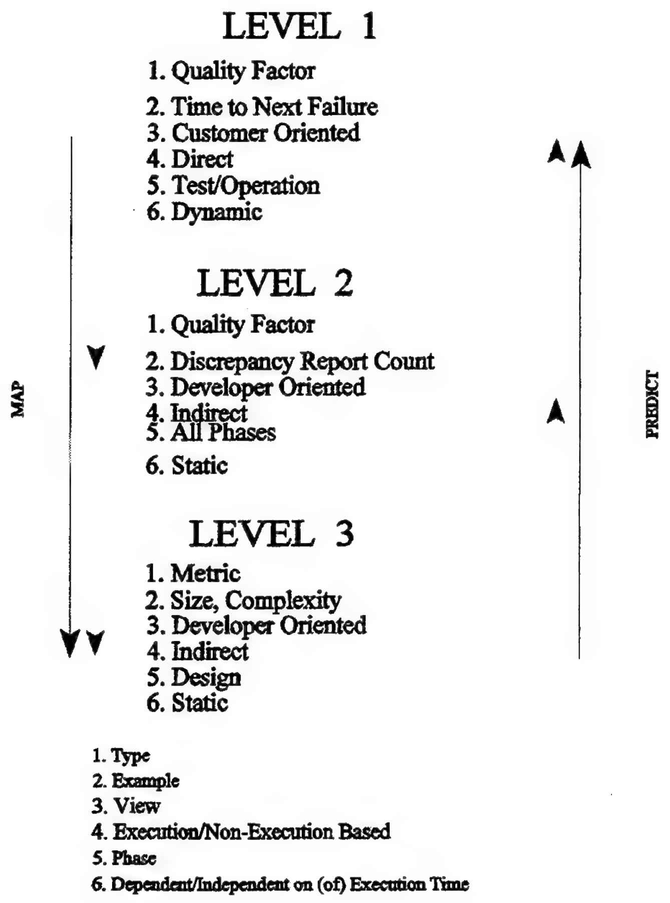


Figure 2: Levels of Measurement

This figure also shows, on the right side, that we want to predict the quality of later phases, using metrics that are available in the early phases. In addition, this figure shows on the left side that we want to map from failures observed in later phases to the metrics of early phases in order to identify the cause of the failures.

Step 3: Collect the Data

Without data, reliability predictions cannot be made. For this data collection, a Data Base Management System (DBMS) would be helpful. For computational purposes, the file management system of certain software reliability tools (e.g. SMERFS and Statgraphics, which are discussed later in the handbook) are usually adequate. However, to manipulate large amounts of failure and metrics data, a specially designed DBMS may be beneficial. This DBMS would allow for data sorting for various analyses and reporting purposes. This is easily accomplished by identifying the key fields of the data (date, time of failure, type of failure, degree of failure) and relating those fields with others. By using the DBMS's query capability, various statistics and reports can be produced by the touch of a few keys. This data can then be properly formatted to be input into the model and further evaluated for trends.

The elements of the database are shown in Table 1.

System ID	Days # (since start of test)	Problem Report ID	Problem Severity	Failure Date	Module with Fault	Description of Problem

Table 1 Failure Data Collection Format

For each system, there should be a brief description of its purpose and functions. The *Days #* field could be noted in *hours* or *minutes*, as appropriate. It is recommended that the *Problem Report ID* field be coded to indicate Software (S) failure, Hardware (H) failure, or People (P) failure.

A more detailed discrepancy report is found in Appendix A. This detailed report could be implemented by the organization as it becomes more familiar with the Software Reliability Process.

Step 4: Establish Problem Severity Levels

The organization will need to establish some consistency in describing the faults it discovers. This will allow better analysis and classification of failures in the analysis and reliability predictions. Some recommended severity level descriptions are as follows:

Level 1. Loss of life, loss of mission, abort mission

Level 2. Degradation in performance

Level 3. Operator annoyance

Level 4. System ok, but documentation in error

Level 5. Error in classifying a problem (i.e., no problem existed in the first place)

Note: Not all faults result in failures.

These levels should be recorded as part of Table 1.

Step 5: Estimate Model Parameters

Once a model has been chosen to be applicable to a particular system, the necessary model parameters must be estimated, using SMERFS. For the purposes of this project, the Schneidewind Software Reliability Model is being used. Three parameters are used in this model and will be used for MCTSSA: α, which is the failure rate at the beginning of the testing interval "s", β, which is the failure rate per failure, and "s," the first interval used in parameter estimation. These parameters are discussed further later in the handbook.

Step 6: Select the Optimal Set of Failure Data

This stage selects the subset of failure data, starting with the beginning interval, "s" through "t," the last observed interval, that will give the best parameter estimates and the most accurate predictions. It relies on the observation that both the software process and product change over time. Therefore old data may no longer be representative of the current and future state of the process and product and , therefore, not as applicable for reliability prediction as the more recent data. This step is discussed in detail later in the handbook.

Step 7: Identify the Operational Profile

The operational profile describes the system's environment. It is usually discussed in terms of modes (single node or multi node operation), frequency of use of a particular station with each station performing a different function (e.g. Workstation 1 performing database functions, Workstation 2 performing word processing functions), and the frequency of function execution (the amount of time the application has been running). It includes the input variables (e.g., a listing of available equipment or a ship's destination), the functional environment of the program (i.e., a specific function the system is to perform such as sorting the available equipment by minor property number), and the output variable (e.g., a printout of the ship's destinations for the next two months). In this framework, a failure can be seen as a departure of the output variable from what it is expected to be. (Musa, 1987). In the *Shuttle* example, it is appropriate to use a single software system (i.e., single node). The applicability of the Schneidewind Software Reliability Model to Marine Corps multi-node systems is discussed in Section 5 on page 60. A description of the attributes of this environment can be found in that section of the handbook.

As part of the operational profile, the organization would be using the obtained failure data and calculating the various parameter inputs to be used in the reliability model.

Step 8: Make Reliability Predictions

This step is the key to predicting the reliability of the software under evaluation. Each of the listed predictions and the applicability to a managerial decision is described in detail in the Basic

Concepts section of the handbook, starting on page 17. The possible predictions resulting from the model application are:

- Time to Next Failure
- Cumulative Failures for a Specified Time
- Remaining Failures and Fraction of Remaining Failures
- Total Failures over the Life of the Software
- Test Time to Achieve Specified Remaining Failures
- Operational Quality

Step 9: Validate the Model

This step evaluates the model to determine if it actually measures what the model is designed to measure. The predicted values are compared to the actual values to make a determination of the model's validity. As an example, if the model predicts the time to next failure will be two periods, this predicted time would be compared to the actual time. Validation is achieved after certain numbers and types of predictions have been made with a specified accuracy (e.g., average relative error of $\leq$ 20%).

If, however, the values do not compare favorably, the data used in the model should be carefully examined to identify if anything unusual can be found. If the data appears valid, and the model prediction does not match reality, different models would need to be investigated. For the purposes of this handbook, the Schneidewind Reliability Model will be used exclusively.

Step 10: Make Reliability Decisions

The purpose of implementing a reliability program is to provide the manager with additional information through which he can make informed decisions. Reliability decisions such as "Is the software safe enough to not cause loss of life or mission?" can be made as a result of the model's predictions. This particular decision can be applied to the *Shuttle* software. Here the manager must decide whether to launch the *Shuttle* based on the software reliability predictions. For this example, the predicted remaining failures must be less than a specified critical value and the predicted time to next failure must be at least as long as the mission duration plus some safety margin. This application will be addressed later in the handbook using numerical examples.

For any organization, the predicted software reliability can be key to the managerial decision to accept final delivery of the product. If the software is *predicted* to perform within specifications, the software can be accepted by the organization as fulfilling the contractual obligations. If it is predicted to fall short of the desired goals, further discussion may be needed in addition to further testing and evaluation.

Step 11: Use Software Reliability Tools

There are software reliability tools available to make the model calculations easier to achieve. The *Statistical Modeling and Estimation of Reliability Functions for Software*, SMERFS, is a software package available for this purpose. (Farr, 1993) A sample SMERFS session is outlined in the Testing Procedures section of the handbook found on page 41.

G. SUMMARY OF SOFTWARE RELIABILITY IMPLEMENTATION PLAN

In summary, the first phase in the software reliability engineering (SRE) process is to state the organization's reliability goals. These goals can be ideal or conceptual but must have some basis in reality. A goal of "0%" defects might be the ideal objective, but it would not occur in the real world. Imagining for the moment that it could happen, it would cost an extraordinarily large sum of money to obtain. (Recall Figure 1, the Software Reliability Tradeoff Curve).

The second phase of the SRE involves testing. It is here that the failure data is collected and formatted for inclusion in the model of choice. The test plan used must be consistent with the goals established. If a goal is to have a maximum number of remaining failures set at less than one, then the test plan must be able to predict the remaining number of failures in the software. The tests provide insight into the future -- what may occur as a result of using this software. This insight is used to either forge ahead with actual implementation of the software or return to the drawing board and reassess the system. It will provide an indication as to whether or not additional testing is needed because the results to date may be inconclusive or show an undesirable trend. The test results also allow the manager to prioritize his assets. It can help him to decide where he should assign his resources. Is Module C predicted to be more reliable than Module B? If this is true, he may decide to allocate the majority of his resources to Module B to improve its reliability.

Software reliability is an iterative process. The organization must continually update its expectations about its software and software reliability. It should not stop with one trial run of the model; it must continue to collect data over long periods of time for each of the systems in use. In light of this, the organization must be constantly looking ahead. As more data is collected over longer periods of testing and operation, this larger data set can be used in a reliability model to make more accurate predictions for longer times into the future. It is an integral part of the SRE to have the data stored and available in a data repository.

These steps provide the reader with the general overview of the reliability methodology that should be carried out as part of the software reliability engineering (SRE) process. The next section provides amplifying information regarding the data that must be collected, how it is analyzed by the model, and how the results of the model can be interpreted.

This page intentionally blank.

SECTION 2: BASIC CONCEPTS USED IN THE SCHNEIDEWIND MODEL

In the previous section, this handbook presented an overview of the SRE process by briefly introducing its key components. This section will further discuss software reliability predictions the Schneidewind Model produces as a result of the data collected by the organization. Applications of the usefulness of these predictions are briefly described. Specifically, this section gives the manager additional information on the mathematical foundations of software reliability engineering. The mechanisms MCTSSA can employ to calculate these predictions can be found in Section 3, Testing Methodologies, on page 39.

A. INTRODUCTION

Data collection must be started at the design and developmental phases of the process including any failure data obtained from the developer-run tests. Data obtained from these early stages can then be used during the independent verification and validation phases to predict the software's reliability. However, this data collection would not stop at the development phase; data should be collected throughout field operations. Data obtained at this stage can be used for future software design projects and could lend itself to further model validation.

As discussed in the earlier sections of this manual, a model is only able to make **predictions** regarding the reliability of the software. These predictions can be used as a management aid for resource allocation and identifying the need for additional testing. Tests evaluate how reliable the software is. They measure how well the software performs compared to the desired performance levels stated by management in the design specifications.

Modeling allows the manager to get a "feel" for how well the software will perform based on actual data. This permits him to "look into the future" and predict how well the software will perform a week from now, a month from now, a year from now. . . The Schneidewind Software Reliability Model addresses the optimal selection of actual test data to be used in making software reliability predictions. The following sections describe the basic concepts used in this model and their implications for management. Numerous examples from the space *Shuttle* will be used because of the abundance of available test data . Where applicable, MCTSSA examples will additionally be discussed.

Although an abstract discussion of the model may help some individuals understand its applicability, the following scenario is proposed to give the practitioner an understanding of the model application and the uses for the application results. Try to keep this scenario in mind as each of the model components and predictions is discussed. The scenario will be revisited in the application section of the handbook for further discussion.

B. SCENARIO

A manager must decide whether or not to launch the space *Shuttle* for a mission expected to last ten days. He has collected failure data on the software to be used in the launch and has input the data into the model. Based on his confidence in the model, and the predictions made by the model, he will make his decision to launch or not.

C. PREDICTIONS

The following predictions can be made by the Schneidewind Software Reliability Model:

- Time to Next Failure
- Cumulative Failures for a Specified Time
- Remaining Failures and Fraction of Remaining Failures
- Total Failures over the Life of the Software
- Test Time to Achieve Specified Remaining Failures
- Operational Quality

Each prediction and its managerial applications are discussed in the following sections.

1. TIME TO NEXT FAILURE

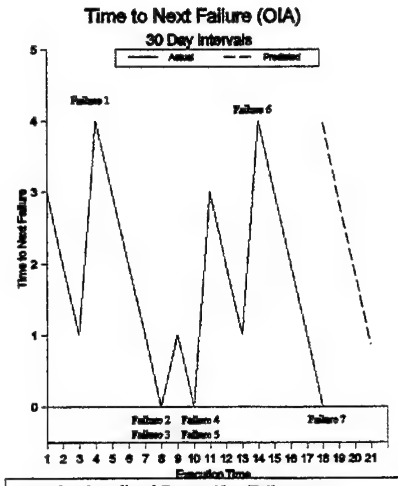
(a) RATIONALE

The following section discusses the significance of time to next failure calculations as it relates to software reliability predictions. This information is important for the manager in that it permits him to make an informed, educated decision on the reliability of the software. As a simplistic example, if the predicted time to next failure is three days, but the software is scheduled to be run for ten days, the manager can anticipate that a failure will occur before the mission is complete. He must then decide whether or not he wants to take that risk.

(b) DEFINITION AND CALCULATION OF TIME TO NEXT FAILURE

The time to next failure can be described as the amount of time that will elapse from the present time, t, until the next recorded failure occurs. In other words, it is the predicted amount of time it will take for the next failure to occur. Execution time is measured from the beginning of a test. This execution time is recorded in convenient intervals of time. As an example, a convenient interval of time for the *Shuttle* program is 30 days. This will be seen on the graphs displaying calculations of time to next failure. However, an organization can set its own interval. In some MCTSSA examples, an appropriate interval would be one week (five workdays).

Figure 3 is a tool that can be used as a management aid. It shows the predicted and actual times to next failure for current execution times. The graph can be read in the following way. If we take a given failure, Failure 1, for example, it occurs at t = 4 (read from the x-axis); therefore, at t = 1, the time to next failure will be equal to 3 (read from the y-axis), (4 - 1 = 3). At t = 2, the time to next failure will be equal to 2, (4 - 2 = 2). At t = 4, Failure 1 occurs, so the time to next failure is 4, (8 - 4 = 4). In this figure, we **predict** the time to next failure to be 4 (at t=18) for Operational Increment A (OIA) on the **dashed curve**, where an Operational Increment is the software system that flies in the *Shuttle*. This curve is derived from additional information and testing (using the Schneidewind Model). Table 2 shows the failure data that was used to construct the **actual** part of Figure 3.



Actual and predicted *Time to Next Failure* are obtained for given *Execution Time*. Failures were observed in intervals 1-18 and predicted in 18-21.

Figure 3: Time to Next Failure

Time Interval	**Failure Identification Number**	**Time to *Next* Failure**
1	--	3
2	--	2
3	--	1
4	1	4
5	--	3
6	--	2
7	--	1
8	2, 3	0
9	--	1
10	4, 5	0
11	--	3
12	--	2
13	--	1
14	6	4
15	--	3
16	--	2
17	--	1
18	7	--

Table 2. Data Used to Construct Time to Next Failure Graph

(c) SCENARIO REVISITED

With the *Shuttle* mission scheduled to last ten days, the ideal situation regarding time to next failure would be to have the next predicted failure occur at a period of time greater than the mission length. In this situation, the next failure should be predicted to occur after the *Shuttle* has safely returned home, i.e., the time to next failure should be greater than ten days. Although this is a simplistic approach, and does not include other factors, it can give the manager some quick information about the reliability of his software. Other predictions should be included in the decision process. These predictions are discussed in the following sections.

2. CUMULATIVE FAILURES

(a) DEFINITION AND APPLICATION

Cumulative failures are the total failures predicted to occur at a specific point of time in the future. The benefit of this prediction is that it can be used to anticipate the total failures, for a given execution time, and help the manager prepare to deal with them. Also, if the predicted number of failures is considered unacceptable, the software and its processes can be investigated to see where the problems lie.

3. REMAINING FAILURES, (R), AND FRACTION OF REMAINING FAILURES, (p)

(a) RATIONALE

The *number of remaining failures* provides the manager with valuable information about the reliability of his software. Specifically, it gives him an indication of the software's reliability by predicting the remaining failures (undiscovered failures) that still exist in the software. With this information, he can make an informed decision as to whether the software meets his requirements.

If the number of remaining failures is high, the software will typically not satisfy the reliability requirements.

The *fraction of remaining failures* can be used as both a program quality goal in predicting test time requirements and , conversely, as a indicator of program quality as a function of test time expended.

(b) DEFINITION AND CALCULATION OF NUMBER OF REMAINING FAILURES, (R)

The *number of remaining failures* is measured from a given interval and identifies the predicted count of failures remaining in the software. If one predicts the total number of failures that will occur in the software, the remaining failures can be predicted though simple subtraction: total number of failures minus the number of failures found to date. The *fraction of remaining failures, p,* is calculated by taking the number of remaining failures and dividing that number by the total failures predicted for the software.

(c) APPLICATIONS

Management will set guidelines on the desired value for R. Normally, R is set to be less than one. This means that the *expected* number of remaining failures that will occur from the present time to the end of the software execution cycle (also known as run time or "mission time") should be less than one. If the predicted value for R is greater than one, this indicates that the software could contain remaining faults and failures that are unacceptable. If the system is mission critical or has the potential to cause harm to human life, the prediction of $R > 1$ should tell the manager that there would be serious risk if he uses the software as it is currently designed.

(d) SCENARIO REVISITED

With the *Shuttle* mission scheduled to last ten days, and with a prediction of time to next failure of four 30 day intervals (see page 20) coupled with a prediction of **R**<1, the manager would have confidence that the software would operate reliably during the mission. If on the other hand, one or both of these predictions do not meet the thresholds, the manager should seriously consider postponing the launch.

4. NUMBER OF FAILURES REMAINING IN ONE MORE TEST PERIOD

(a) DISCUSSION AND APPLICATION

The number of failures remaining in one more test period gives the manager information about the reliability of the software during that particular time interval. This information can prompt the manager to continue testing or to deploy the software, provided that the time to next failure and predicted number of remaining failures are acceptable. A test period of thirty days of **execution time** can be used, as is done in the *Shuttle* software; or it can be a **calendar time** of one work-week (5 days), as is done in LOGAIS; or any other convenient measure of time.

If the manager must make a decision whether to deploy the software and discontinue testing, he will look for an acceptable value for the predicted number of failures remaining in one more test period. Normally, this number should be significantly less than one. The ideal figure for this calculation would be close to zero, e. g., .0001. If the value is close enough to zero for the manager, he may decide to take the risk, discontinue testing, and deploy the software.

5. TEST TIME NEEDED TO ACHIEVE DESIRED RELIABILITY LEVEL

(a) DISCUSSION

This information provides the manager with an estimate of the amount of time needed for software testing to achieve a given level of reliability, similar to time needed to obtain "fault free" software. This calculation is based on two key computations: the fraction of remaining failures, "p," and the predicted maximum number of failures over the life of the software, which was previously described (see page 23).

(b) CALCULATIONS

(1) Maximum Failures

The predicted maximum number of failures over the life of the software ($T=\infty$) is defined as: $F(\infty)=\alpha/\beta+X_{s-1}$, where X_{s-1} is the failure count in the range 1,s-1 (i.e., incuding the first failure count interval and up to and including the interval prior to interval "s".

The benefit of this prediction is that it provides an indication of the total failures and faults that will occur over the life of the software. Thus the software manager can be alerted during test that there could be problems with the software during operation. Also, total failures are used in the prediction of remaining failures.

(2) Remaining Failures and Fraction of Remaining Failures

The predicted number of remaining failures is: $R(t)=(\alpha/\beta)-X_{s,t}=F(\infty)-X_t$, where $X_{s,t}$ is the observed failure count in the range s,t and X_t is the observed failure count in the range 1,t, where "t" is the last observed failure count interval. As already mentioned, the benefit of this prediction is that it may indicate residual or remaining problems with the software. Furthermore, *fraction of remaining* *failures,* (p=R(t)/F(∞)), can be used as both a *program quality* goal in predicting *test time requirements* and, conversely, as an indicator of *program quality* as a function of *test time* expended.

(c) APPLICATION

Figure 4 provides an example of the *Shuttle Primary Avionics Software* entity designated *OIA* and illustrates how *p* might behave as increased *test time* is applied (represented by "test intervals"). From this type of information a program manager can determine whether more testing is warranted, or whether the software is sufficiently tested to allow its release or unrestricted use. Note that required test time rises very rapidly at small values of **p** and **R(t)**. ***Note***: You should read the test time from the left axis as a function of p, and read the remaining failures from the right axis, as a function of p. Do not combine a value from the test time axis with a value from the remaining failures axis.

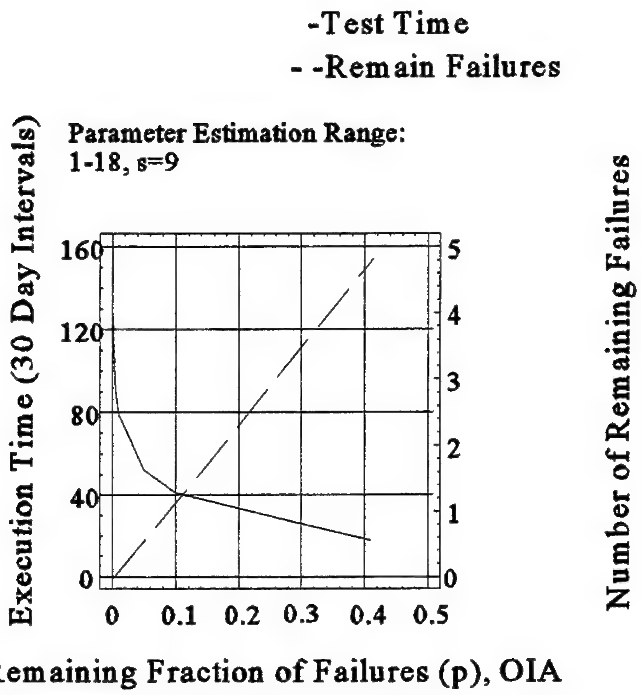


Figure 4: Test Time for Given Remaining Failures

6. MEAN SQUARE ERROR (MSE)

(a) APPLICATION

This section is included here for continuity purposes in discussing the components of the Schneidewind model. Although MSE is not a "prediction" as are the other numerical calculations previously discussed, its determination is key to the success of the model. It is an important statistical value that must be calculated to determine the correct numerical inputs for the model.

Data used in the model is collected from the beginning of the project cycle. However, the software and process used in the software development can change over time. Old data may not have the same relevance as it had when it was "new." For this reason, one may want to ignore "old" data in favor of "new" or more recent data. It may be possible to obtain more accurate predictions of future failures by excluding or giving lower weight to the earlier failure counts. The MSE identifies the time interval where this distinction should be made. There are three types of predictions where MSE can be applied: cumulative failures, time to next failure, and remaining failures.

(b) DEFINITION

The MSE minimizes the sum of the variance and the square of the bias of predicted failures (or time to next failure). It is a statistic that computes the sum of the squared differences between model predictions and actual cumulative failure counts in the range of s, t. This value is used to select the optimum value of the interval where measurements will begin. The following sections describe the computations needed for calculation of MSE. They should be read by the interested reader who desires a mathematical understanding of the calculation process. Other readers may proceed to page 32, Test Time to Achieve Desired Specified Remaining Failures.

(1) **Mean Square Error Criterion for Cumulative Failures**

The Mean Square Error (MSE_F) criterion for cumulative failures is used to select the optimal value of s (i.e., the value of s that results in the minimum value of MSE_F). The result is an optimal triple (β, α, s). The MSE_F computes the sum of the squared differences between model predictions and actual cumulative failure counts $X_{s,i}$ in the range $s \le i \le t$, where $X_{s,i}=X_i-X_{s-1}$.

$$MSE_F=\frac{\sum_{i=s}^{t}[\alpha/\beta(1-\exp(-\beta(i-s+1)))-X_{s,i}]^2}{t-s+1}$$

Figure 5 shows an example of MSE_F in both the parameter estimation range 1,20 (MSE_F computed prior to prediction) and the prediction range 21,30 (MSE_F computed after prediction). Because the latter MSE_F is a minimum at s=11 -- the same as the former -- it confirms that s=11 would have been the best interval to start using the failure data.

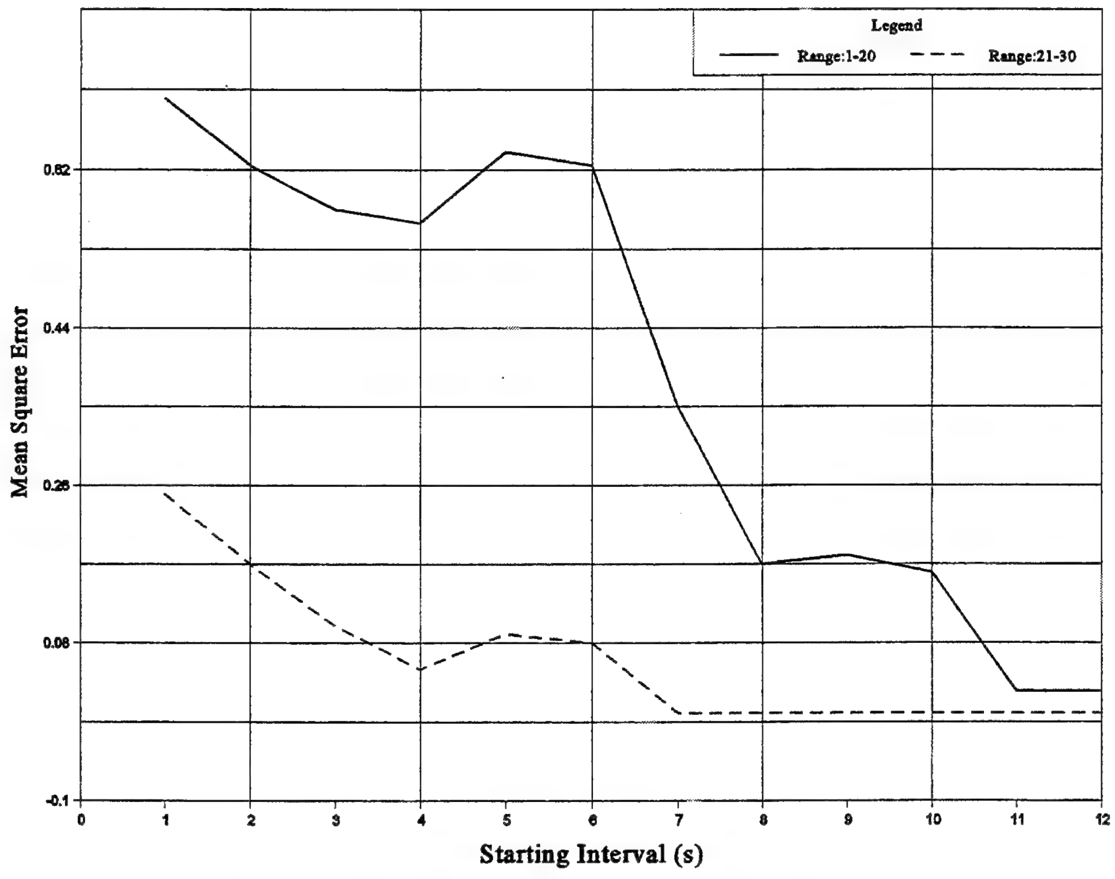


Figure 5: Prediction 21-30. Parameter Estimation 1-20

(2) **Mean Square Error Criterion for Time to Next Failure(s)**

The Mean Square Error (MSE_T) criterion for time to next failure(s) is defined similarly and is given by:

$$MSE_T = \frac{\sum_{i=s}^{J-1} [[\log[\alpha/(\alpha-\beta(X_{s,i}+F_{ij}))]/\beta-(i-s+1)]-T_{ij}]^2}{(J-s)}$$

for $(\alpha/\beta)>(X_{s,i}+F_{ij})$

The terms in MSE_T have the following definitions:

i:	Current interval;
j:	Next interval j>i where $F_{ij}>0$;
X_s,i:	Cumulative number of failures observed in the range s,i;
F_{ij}:	Number of failures observed during j since i;
T_{ij}:	Time since i to observe number of failures F_{ij} during j (i.e., T_{ij}=j-i)
t:	Upper limit on parameter estimation range; and
J:	Maximum j≤t where $F_{ij}>0$.

Figure 6 shows both MSE_T and Mean Relative Error ($MRE=\Sigma_i(|X_i-F_i|/X_i)/N$ for N intervals) versus *s* for the post-prediction range. The same MSE_T result was obtained for the observed range. In this case, s=5 was identified as best prior to prediction but s=6 turned out to be best after prediction.

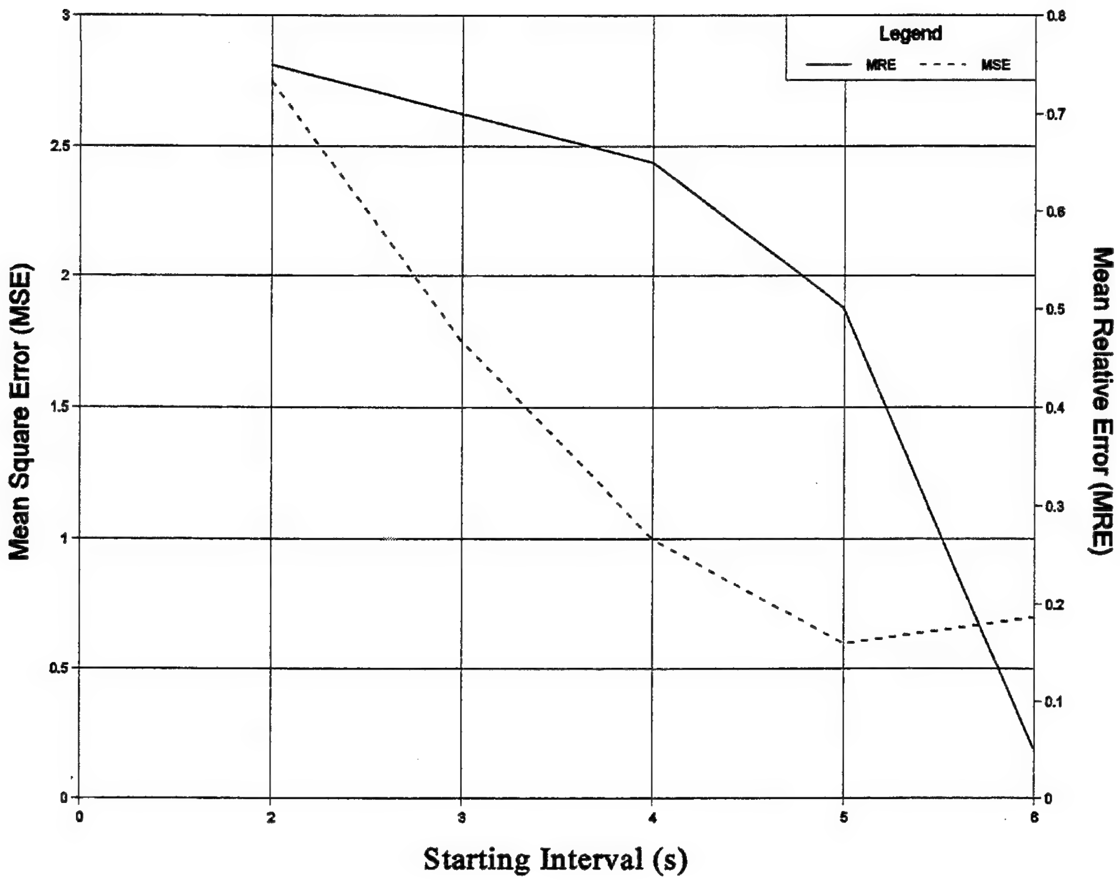


Figure 6: MSE and MRE: Time to Failure

(3) **Mean Square Error Criterion for Remaining Failures**

The Mean Square Error (MSE_R) criterion for number of remaining failures is given by:

$$MSE_R = \frac{\sum_{i=s}^{t} [F(i)-X_i]^2}{t-s+1}$$

where F(i) is the predicted cumulative failures at time i and X_i is the cumulative observed failures at time i.

It should be noted that parameter estimates and MSE evaluations are model setup operations -- not predictions of the future. Rather, during setup, the model is tuned to obtain the best estimates of the parameters by making the best fit of the model to the observed failure data (MSE). Once this has been accomplished, the model is ready to be used for future predictions.

7. TEST TIME TO ACHIEVE SPECIFIED REMAINING FAILURES

(a) DEFINITION

The predicted test time required to achieve a specified number of remaining failures, where $R(t_2)$ is the specified number of remaining failures at t_2, is:

$$t_2=[\log[\alpha/(\beta[R(t_2)])]]/\beta+(s-1)$$

(b) APPLICATION

This concept is shown in Figure 7 for *OIA*, where *remaining failures*=.6 at $\mathbf{t_2}$=52 is marked. This value of $\mathbf{t_2}$ also is in the region of the graph where further increases in $\mathbf{t_2}$ would not result in a significant increase in reliability. The value of this prediction is that software managers can: 1) plan for the amount of test time necessary to achieve a specified reliability goal and 2) determine whether the reliability goal will be achieved with a given amount of test time.

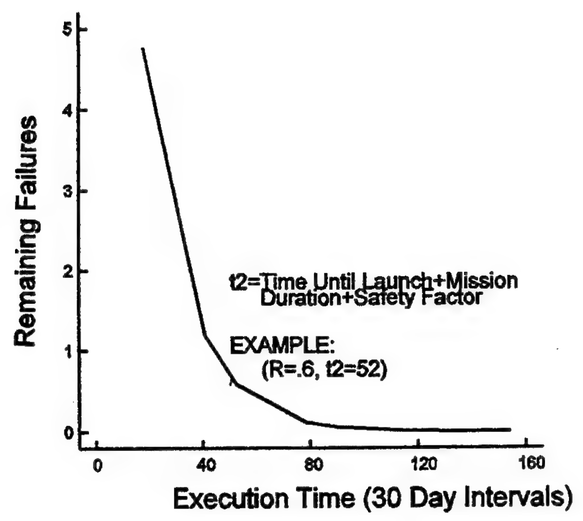


Figure 7: Remaining Failures vs. Test Time

Another type of analysis that can be made with *test time* is shown in Figure 8 where $\mathbf{t_2}$ is plotted as a function of **p** for three modules. The benefit of this prediction is that the software manager can predict how much test time should be allocated to each module to achieve a given level of reliability, as specified by **p**. For example, in Figure 8, for a given **p**, *Module 3* will require the most test time. Conversely, for a given $\mathbf{t_2}$, this module will have the worst reliability (SCH 92).

These figures can be used as management decision tools. The graphical representations of test time predictions provide the manager with valuable information. He can use this information to

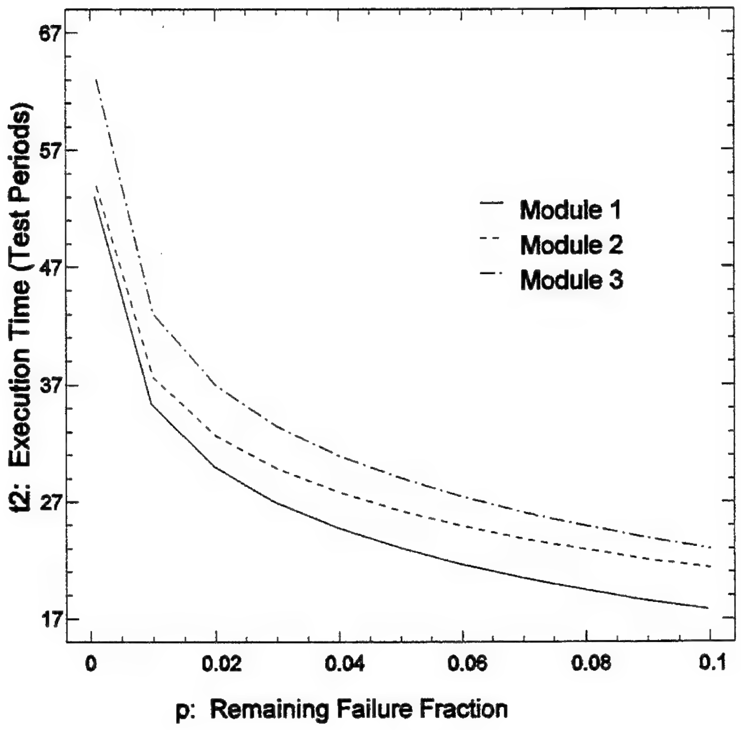


Figure 8: Execution Time to Reach Fraction of Remaining Failures

allocate his resources to include additional test time and personnel. These decisions will be based on his priorities and the predicted software reliability.

8. TEST TIME NEEDED TO OBTAIN "FAULT FREE" SOFTWARE

(a) DISCUSSION

"Fault Free" software can be described as software where the remaining number of failures over the life of the software is, for practical purposes, "zero," (e. g. .0001). There would be no failures remaining in the software. The predicted test time required to achieve a specified number of remaining failures is calculated through the Schneidewind model.

(b) APPLICATIONS

This value can provide management with an approximate time value, and hence, dollars, it would take to test the software until there are "zero" failures remaining. He may decide to allocate all his resources to testing this particular piece of software, or he may decide to stop testing and send the software back to the developers for repairs and modifications.

9. OPERATIONAL QUALITY

(a) DEFINITION

The operational quality of software is defined as: **Q=1-p** (Where "p" was defined as the fraction of remaining failures).

This equation is a useful measure of the *operational quality* of software because it measures the degree to which faults have been removed from the software, relative to predicted *total failures*. Operational Quality is plotted against Execution Time in Figure 9. We again observe the asymptotic nature of the reliability-testing relationship in the great amount of testing required to achieve high levels of quality.

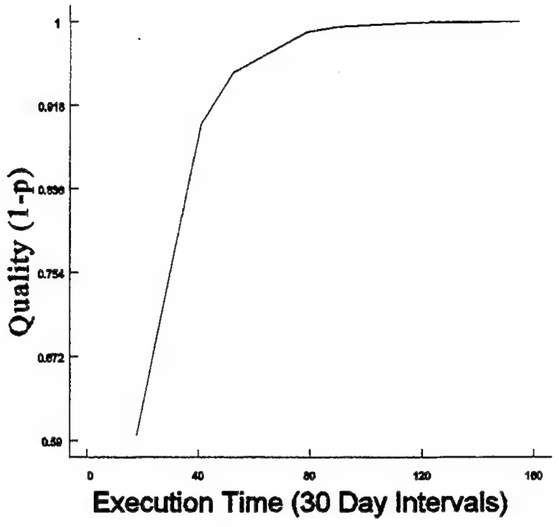


Figure 9: Quality versus Test Time

(b) APPLICATION

When management is provided with this information, it can make trade-off decisions regarding quality and cost (inspection time). Higher quality will require more inspection time. The converse is also true. The manager can inspect the trade-off curve and decide where he receives the best gains for his investment. The curve will eventually show decreasing marginal gains.

D. SUMMARY

This section provided some background information on the types of predictions available by employing the Schneidewind Software Reliability Model. It also gave managerial applications for use of the predictions. Key to this section was the data. For without data, no predictions would be possible. It cannot be emphasized enough how important it is to collect data as early in the development process as possible.

The next section will discuss how an organization can make the predictions discussed in Section 2 by using certain software packages. Additionally, application of these predictions to the *Shuttle* program will be discussed.

This page intentionally blank.

SECTION 3: TESTING METHODOLOGIES EMPLOYED

The following section discusses the three key components to making software reliability predictions. These components include the two software packages that make the necessary calculations easier to compute (SMERFS and Statgraphics) and the reliability model itself (Schneidewind Software Reliability Model).

A. SMERFS

Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS), not the little blue men from outer space, is a software reliability modeling tool that can be used to gain insight into the reliability of the software being tested. SMERFS is a tool that implements the models developed by Schneidewind and a number of other software reliability researchers. Using the Schneidewind Model component of SMERFS, two types of predictions can be made: for a given number of time intervals, how many failures will occur? secondly, for a given number of failures, how many time intervals will be required for the failures to occur? After inputing the software failure count data, usually from an input failure data file, the first step is to determine the optimal starting value for "s" as determined by the table of MSE values; usually the "s" with the minimum MSE will be selected.

B. THE SCHNEIDEWIND SOFTWARE RELIABILITY MODEL

As stated above, SMERFS is a statistical software tool that can perform various calculations on an input failure data file to predict both the number of failures and the time to next failure. However, before these calculations can be made with the Schneidewind Software Reliability Model, SMERFS must calculate the Mean Square Error, as previously discussed on page 27 to determine the optimal starting interval, "s," which corresponds to the minimum MSE between predicted and actual values of failure counts or time to failure.

C. STATGRAPHICS

Because SMERFS does not contain all the equations used in the model, we have implemented some equations in Statgraphics (version 5.2 for DOS, which can run under Windows). Statgraphics is a software tool designed to aid in calculations of mathematical formulas and provides statistical analysis and graphing capabilities. This tool is used to predict the required test time to achieve a desired reliability level, using the following formula:

$$t_2 = [\log[\alpha/(\beta[R(t_2)])]]/\beta + (s-1)$$

The values for alpha, beta, and "s" are retrieved from the data collected using SMERFS. In addition, the MSE for remaining failures,

$$MSE_R = \frac{\sum_{i=s}^{t} [F(i)-X_i]^2}{t-s+1}$$

is not implemented in SMERFS but we have implemented it in Statgraphics. Additional equations that are implemented in Statgraphics are the following: Cumulative Failures, Fraction Remaining Failures, and Program Quality.

D. TESTING PROCEDURES

Using SMERFS, one can address the following two objectives: (1) Why and how a reliability model can be used to predict execution time to next failure, and (2) Why and how a reliability model can be used to predict how long the software should be tested in order for it to be "fault free." The following instructions for SMERFS will achieve these objectives.

1. USING SMERFS

Although most of the instructions for SMERFS show up on the computer screen and are self-explanatory, the following amplifying instructions will assist the first-time user in successfully completing his session. See Appendix A to follow along with the SMERFS printout. User inputs are highlighted (in bold print) for ease of use. ***Note***: Calculation results should be rounded to no more than one or two decimal places, because reliability cannot be predicted with greater precision. However, to be consistent with the SMERFS printouts in Appendix A, the results shown in this section will be left as calculated.

a. Once SMERFS is accessed, the first input required from the user will be the name of the file where he would like the SMERFS output (results) stored. As an example, **a:\smerfs1** would store the resulting SMERFS ASCII file on the computer's A-drive if a disk is inserted. This will make data retrieval easier once the session is complete. The user can then access his "output" file via a word processing program, format the data as he wishes, and print the results.

b. The user will then be asked if he would like to store a plot file for later retrieval. The recommended answer for this question is **0**, (zero), meaning "No".

c. SMERFS will next require the failure data type the user will be working with. At this point the user will enter **4**, for the interval failure counts and testing lengths.

d. Now he will be asked to **enter a 1** for the standard SMERFS file input. This should be followed by the name of the file where his sample data is stored, for example, a file name of **oi618.in**. [This sample file contains the number of failures recorded against an operational increment (OI) of the *Shuttle*. This OI consists of a build of various modules in the *Shuttle* software library. There are 18 count intervals in **oi618.in**. Each interval is 30 days of continuous execution time.]

e. This step will ask the user how he would like the input displayed. The recommended response is to **enter a 3**. This entry will show a table of all the data input through the oi618.in file. However, the user may enter a 0 to display a list of his options at this point.

f. Following the display of data, the same question will reappear regarding the input display. This time the user is recommended to **enter 4** to take him to the SMERFS main menu. He will then be asked if he would like to make some new data files. He should **enter a 0** to void the data restore option.

g. He should then **enter 0** to display the listings available at the main menu. This will present him with nine choices. He should select **option 8 (Executions of the models).**

h. Upon this selection, the user will then **enter a 0** to display the available count model options. He should select **option 4 (The Schneidewind Model)**.

i. The next displays will permit the user to see descriptions of the model or the treatment type. For these options, a **0** should be entered for each option unless he desires the descriptions.

j. The next step will be to investigate the "optimum s" from the various count intervals input into the program. A **1** should be entered here. He will then be asked to enter the range over which "s" should be tested. In general, the user should enter the range of the input failure data (i.e., 1,18 for this application). However for this application, we had previously determined that SMERFS could not compute values for MSE for "s" greater than 9. Therefore for this specific example, the user should **enter 1,9**. This entry will display the table of s, beta, alpha, WLS, MSE_F and MSE_T. The last two terms are the mean square error, as a function of "s", for number of failures and time to failure predictions, respectively (ignore the "WLS" column).

The user should note the table results and select those values for "s" which give him the **smallest** MSE_F and MSE_T.

TIME TO FAILURE PREDICTIONS

k. After the user is comfortable with the data presented in the table, he should enter 0 to conclude the table presentation. He will then be asked to enter the desired model treatment number. He should enter **2**. For the number of associated values of "s" he should enter the corresponding "s" value that gives the smallest MSE_T, for time to failure prediction. In this example, the minimum value for MSE_T is seen for "s" equal to 5. A **5** should be entered. This entry will result in a display of model estimates. Of note in this display should be the *total number of failures*, and the *remaining number of failures*. (Total number of failures: .11722E+02; Remaining number of failures: .17221E+01). These values, as discussed previously, provide the manager with information regarding the reliability of the software he is testing. **He should record these values** for future use in this demonstration.

l. The user will then be prompted to select from two options regarding future predictions. For the sample run, he should select 2 for the prediction of the number of periods needed to discover the next "M" failures. This will allow him to determine the value of "M". He should **enter a 1**. The result will predict the number of *additional* test periods required to discover one more failure. A result of 6.3443 periods results (190.32 days). This implies that the time to next failure, from the present time, will be 190 days.

m. When asked to enter a value of M, the user should **enter 0**. The user will be prompted again to **enter a 0** to end the current predictions.

NUMBER OF FAILURES PREDICTIONS

n. This step moves the user into predicting the number of failures that will occur in one more test period. He will be prompted to enter the model treatment number. He should **enter a 2**.

o. He will then be prompted to enter the associated value of "s" he would like to investigate. He should enter the "s" value corresponding to the minimum value for MSE_F he recorded earlier. For this example, the **value of 6** should be entered. This entry will produce a listing similar to the listing produced in step j. As in step j the key values obtained here are the *total number of failures*, the number corresponding to *plus those skipped*, and the *number of failures remaining*. If the value for *plus those skipped* is not equal to zero, this value must be added to the total number of failures and the number of failures remaining. **The user should record these values**. The example values correspond to

Total number of failures:	14.363
Plus those skipped:	3
# of failures remaining:	4.3626

p. The program will present the user with two options for data evaluation. He should **choose option 1** for the number of failures expected in the next testing period. He will be prompted to enter the number of periods to examine. He should **enter a 1**. This will display the number of failures expected. For this example, it will be .36888. This implies that the number of remaining failures occurring in the next execution cycle (30 days) will be .37. This is the final SMERFS calculation.

q. The user can exit the program by entering the following values in sequence: 0 to end period to examine, 0 to end predictions, followed by a 4 to terminate the model execution, 0 to conclude analysis of model fit, 0 for count model options, 6 to return to the main menu, 0 for a list of main module options, and finally, 9 to stop execution of SMERFS.

(1). **INTERPRETING SMERFS RESULTS**

Using the sample file and the SMERFS software, the following results were achieved:

Time to Failure Data ("s" = 5):

Time to next failure (from present time): 6.34 periods (190 days)
Number of remaining failures (from present time): 1.72
Total number of failures: 11.7
Calculated *fraction of remaining failures*: .15
Time necessary to reduce the remaining failures to .0001: 71.76 periods (5.9 years)

Number of Failures Data ("s" = 6):

Number of remaining failures (from present time): 7.36
Total number of failures: 17.36
Calculated *fraction of remaining failures*: .42
Number of failures that will occur in one more period: .37

Note: Because in this example s=5 was optimal for *time to failure* predictions and s=6 was optimal for *number of failures* predictions, different results are obtained for *number of remaining failures* and *total number of failures*. Because MSE_F applies to failure count quantities like these, the values obtained for s=6 should be used in this example (i.e., *number of remaining failures*=7.36 and *total number of failures*=17.36).

These results provide the manager with useful information regarding the reliabilty of his software, provided he looks at all the data as complementary information. He should not make a decision based on only one piece of the above information, rather, he needs to look at the data in its entirety.

(2). SCENARIO REVISITED

A manager must decide whether or not to launch the space *Shuttle* for a mission to last ten days. He has collected failure data on the software to be used in the launch and has input the data into the model as described in the above sections.

Looking at the data in its entirety, he should **not launch** the *Shuttle*. Even though the time to next failure is predicted at 190 days and only .37 failures are predicted for the next interval (30 days giving the mission a cushion of 20 days), the predicted number of remaining failures is 7.36. This is a significantly high number. (As discussed previously, the manager desires this number to be less than one.) The time to make the software virtually failure free is 72 periods, a long time! The decision must be based on the available model evidence, his confidence in the model, his risk aversion, and any other factors at his disposal. Using only the data from this analysis, the overriding factor of 7.36 possibly life-threatening remaining failures, the manager should not launch the *Shuttle*.

2. USING STATGRAPHICS

Statgraphics is used to augment the reliability predictions obtained from SMERFS. Equations, like the one for t_2 below, can be created using the Statgraphics equation editor feature. Of particular interest in this phase of the predictions is the formula for computing the test time required to achieve a given reliability level. As discussed in earlier sections of this handbook, this amount of test time is defined by the following equation:

$$t_2 = [\log[\alpha/(\beta\,[R(t_2)])]]/\beta + (s-1)$$

Based on the way this equation is implemented in Statgraphics, the user must first calculate *p*, the fraction of remaining failures, for each of the desired number of remaining failures, $R(t_2)$. For this example, $R(t_2)$ will be one, two, three, and four.

a. Once Statgraphics has been accessed, the user will be presented with a menu showing various options for calculations and presentations. He will **depress the F8** function key which will cause a new screen to be superimposed on the menu. Here, he will type "**exec**" for the execution screen to appear.

b. Once the blank screen appears, he should **type t2** at the colon prompt if he wants to see the equation before he uses it in a calculation. Otherwise, he can skip this step. This will display the above t_2 equation which has already been preloaded for the user. For Statgraphics to calculate the numerical value for this equation, the user must input the values for alpha, beta, Xs, s and p. The alpha, beta, and s values correspond to the values obtained from the SMERFS session for the smallest MSE_F value. The Xs value is the number of failures observed prior to s=6 from the same

SMERFS session ("plus those skipped" in SMERFS); the p value is the desired number for the fraction of remaining failures for remaining failures of one, two, three, and four.

c. The user will now enter the above mentioned values in the following format for one remaining failure:

```
alpha GETS .73825
beta GETS .051401
Xs GETS 3
s GETS 6
p GETS (1/(EVAL Ft))
EVAL t2
p
```

These commands will display the value for the test time required to achieve a given reliability level. For this input, the predicted test time required to achieve the reliability level of having one remaining failure is 56.84 thirty day intervals. This will correspond to a fraction of remaining failures equal to .0575952. For the remaining failures equal to two, three, and four the following commands must be entered:

```
p GETS (2/(EVAL Ft))
EVAL t2                          Results:  t2 is 43.35
p                                           p is .115

p GETS (3/(EVAL Ft))
EVAL t2                          Results:  t2 is 35.47
p                                           p is .173

p GETS (4/(EVAL Ft))
EVAL t2                          Results:  t2 is 29.87
p                                           p is .230
```

The above results could be plotted to compare the effect that changing the remaining failures has on the amount of test time needed to achieve that end. An asymptotic relationship is seen between t2 and the fraction of remaining failures, p. Figure 10 is a sample graph that could be obtained.

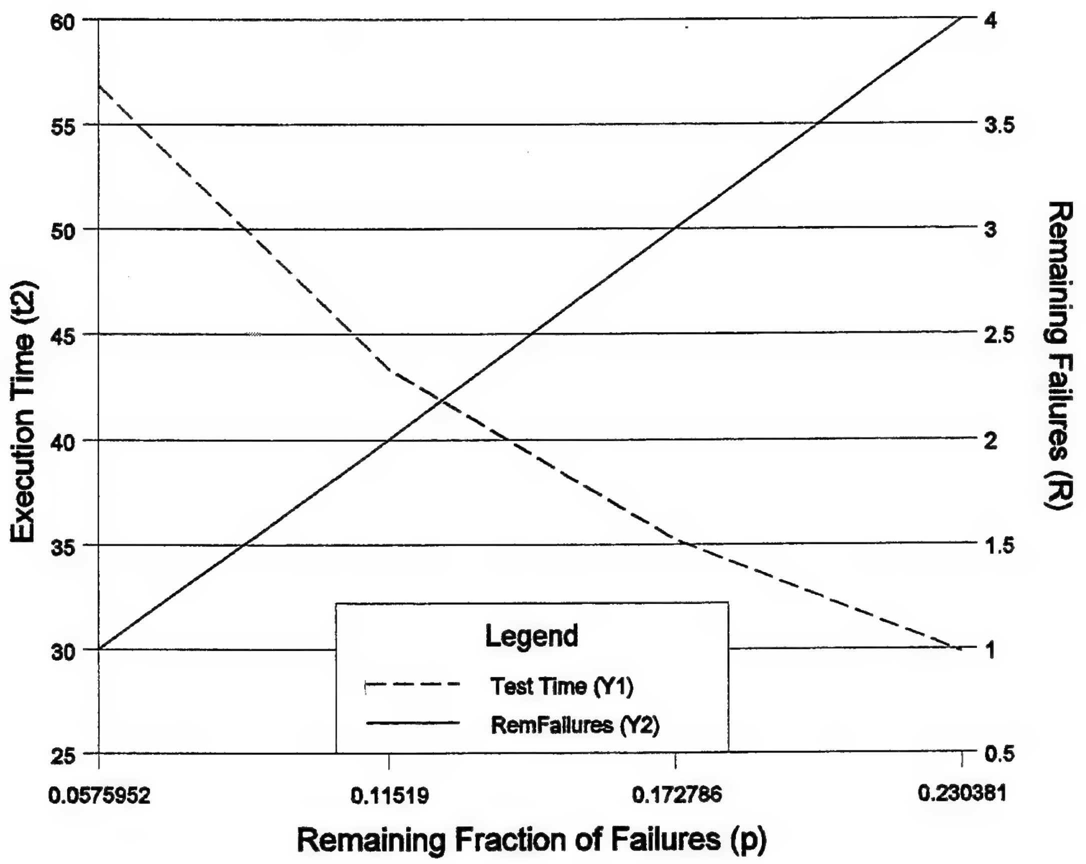


Figure 10: t2 and R versus Remaining Fraction of Failures (p)

(1). APPLICATION

With this information, the manager could gain insight into the predicted amount of time it would take to achieve given reliability levels. Using the scenario mentioned previously, as an example, one could see that it is predicted to take almost 57 periods (totaling 4.7 years) from t=0 to reduce the fraction of remaining failures to .058. The test time curve indicates that there will be a point where there are only marginal returns achieved by additional testing.

Looking at the shape of the curves on Figure 10, the software manager must understand that as predicted reliability increases (the number of predicted failures decreases) there will be a significant increase in the amount of testing time needed to achieve those results. There will come a point were the additional cost of testing will result in only minimal gains in reduced software failures. The manager must make the decision whether to stop testing and deploy the software, based on available funding for testing and the desired reliability levels.

E. CONCLUSION

Management must use all resources available to it to come to a sound, information-supported decision. The model is only a tool to help make this decision. The predictions provided by the Schneidewind Software Reliability Model can give management additional information on the predicted reliability of its software. This can be accomplished by both the developer and implementor using the software reliability engineering process that has been described in this handbook. Using appropriate failure data, the predictions can be used to help make an informed reliability decision. However, the final decision must be made by the manager based on all the information he has available to him.

SECTION 4: APPLICATION OF THE SCHNEIDEWIND SOFTWARE RELIABILITY MODEL TO MCTSSA LOGAIS DATA

A. DATA PREPARATION

In the development of this research project, MCTSSA obtained a database of compiled defect data from the contractor. This database, **Software Edge's** *Defect Control Systems for Windows, Version 2.10*, contained all the defect data the developer recorded on the software during its design. As a database, the software provided a query capability which was used extensively to draw out the appropriate data for inclusion in the Schneidewind Software Reliability model. This data was the **number of defects** recorded during an interval, one day, by date the defect was submitted to the database. This data was then formatted chronologically (by a workday, not calendar day since defects were only listed during normal working hours) for inclusion in a table for easy of readability and analysis. This date sequencing permits reliability predictions to be made. All of the 4584 defects from November 11, 1994 through May 17, 1995 are listed in Appendix C. A determination was made to group the data by five day increments (to simulate the typical work week); this is reflected in Appendix C.

In addition to the data not being recorded in the database by true failure date, there are other challenges the LOGAIS database presented: (1) the data was not true failure data because it was not recorded in execution time when failures occur; rather it was recorded by administrative convenience, by batches at the end of the workday; and (2) the data shows large swings in daily defect count (Figure 11) not showing the expected decrease in number of faults as time progresses (recall Figure 1).

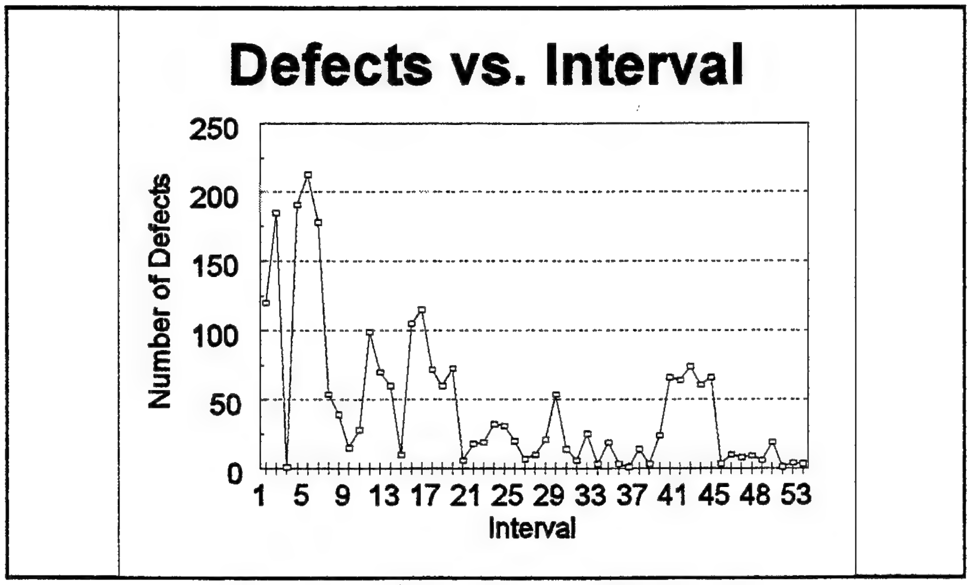


Figure 11. Defect Count vs. Time Interval

If, however, the data is averaged over five day intervals, a decreasing trend does emerge but there are still some unanticipated peaks and valleys. This trend can be seen in Figure 12.

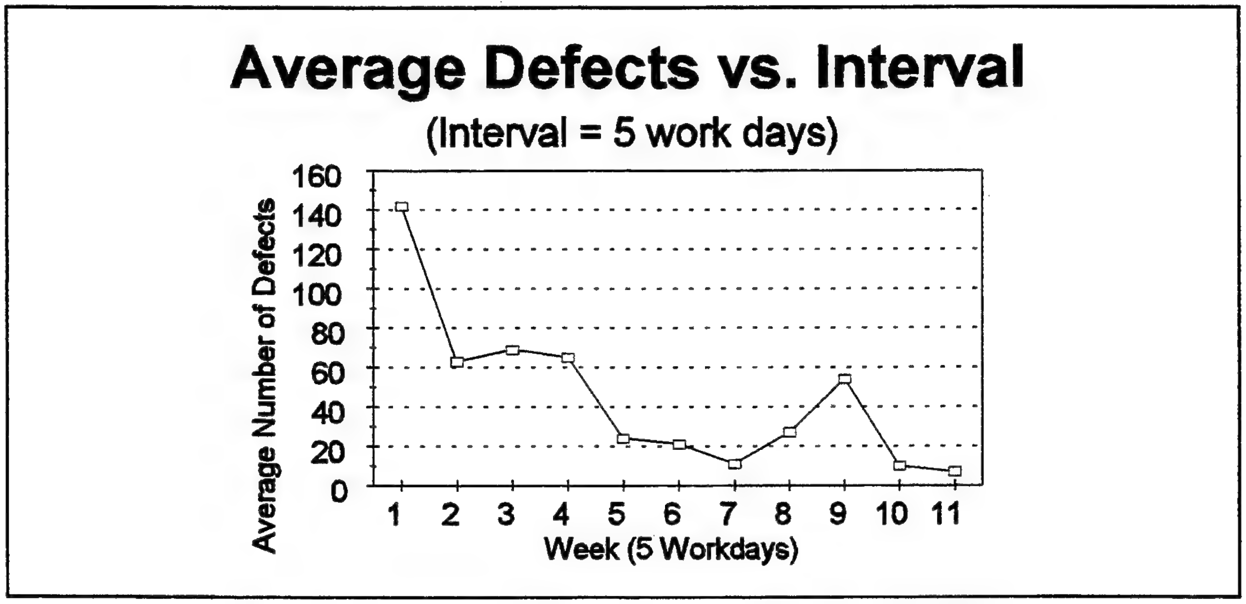


Figure 12. Averaged Defects vs. Typical Five Day Work Week

If the data is smoothed out even further by using a "moving average" of 30 days, a decreasing trend is seen. This is shown in Figure 13.

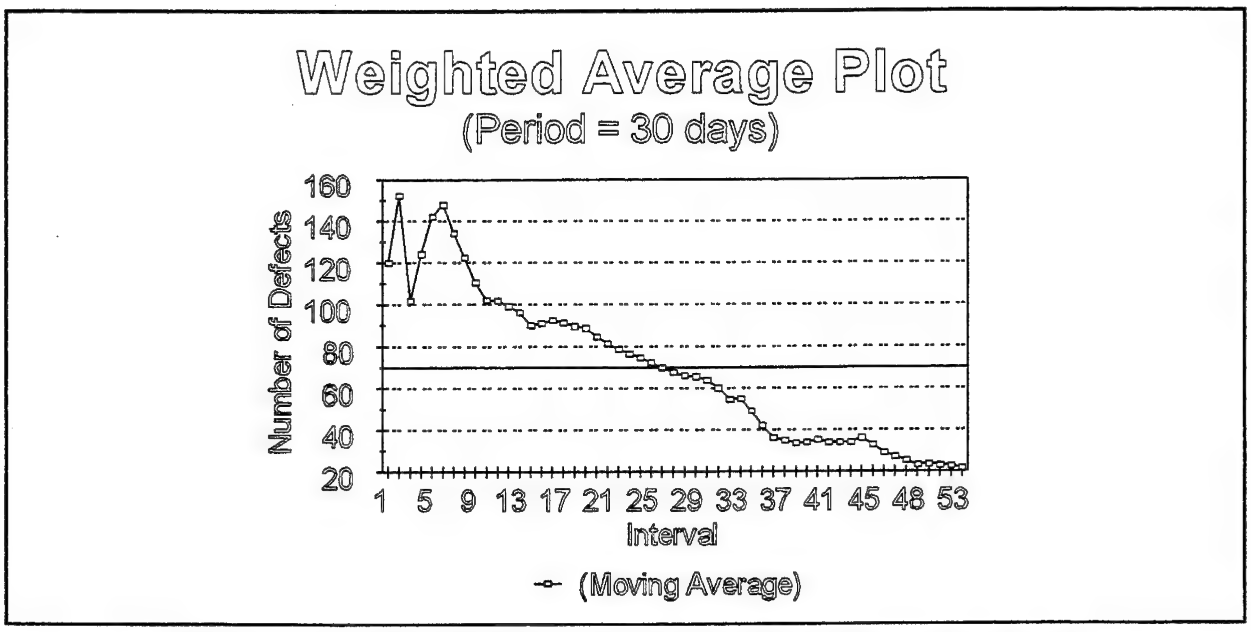


Figure 13. 30 Day Weighted Average of Number of Defects Recorded

These views of the data show the plausibility of using smoothed data as the input to the reliability model instead of using raw data obtained directly from the LOGAIS database. In this test of the data, raw LOGAIS data (the normal approach) did produce fair predictions. Further studies using smoothed data versus raw data would need to be completed and compared to demonstrate if any trends or accuracies are affected by the choice of data used.

This section will discuss the actual application of the Schneidewind Software Reliability Model and the inputs required to obtain meaningful results. **A comprehensive discussion of the model parameters, inputs, and results can be found in previous sections of this handbook.**

B. MODEL APPLICATION AND ITS RESULTS

In order to obtain the most accurate model parameters, both one day and five day intervals were used. By comparing the Mean Square Errors (MSE) of these two intervals, it was seen to favor the five day cycle. Also varied were the length of the input data recorded in the range t = 20, 55 for one day intervals and in the range t = 13,20 for five day intervals, and used the value of the MSE to determine the optimal value for "t."

1. Defect Count Predictions

As previously mentioned, it is advantageous for management to know what the predicted reliability of the software is to help estimate additional testing time needed. This also allows for proper resource management, i.e., assignment or not of a greater number of personnel. This prediction can be achieved through the model's outputs for the predicted number of defects in a selected interval range. This interval range can vary, but is seen as an interval of time in the future, that is, "how many defects are predicted to occur in the next two work weeks?" This was accomplished through the application of the following equation in SMERFS:

$$\mathbf{F(t_1,t_2)=(\alpha/\beta)[1-exp(-\beta(t_2-s+1))]-X_{s,t1}}$$

where this equation is the predicted number of defects in the interval range t_1,t_2; s is the optimal interval to start using defects for the estimation of α and β; and $X_{s,t1}$ is the observed number of defects in the range s,t_1. Here t_1 is defined as t, the end of the parameter estimation range, and t_2 is the prediction interval. The results obtained showed that t = 30 gave relatively good predictions as can be seen in Figure 14.

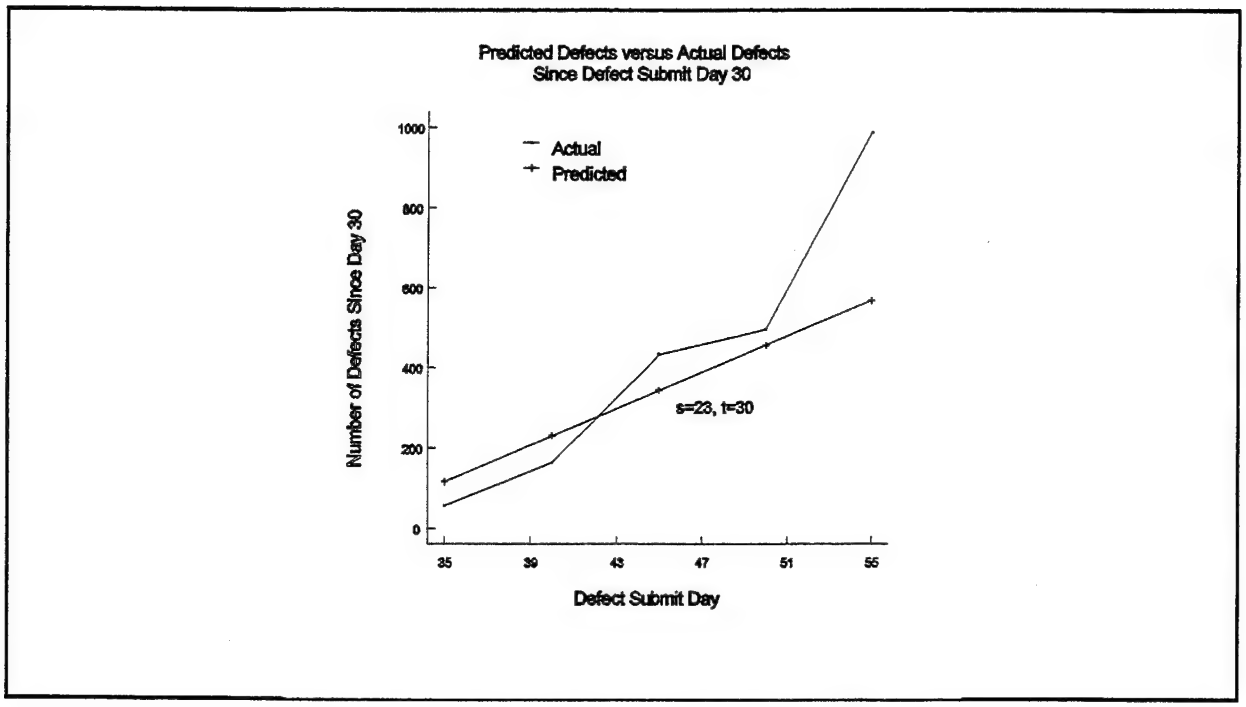


Figure 14. Predicted Defects vs Actual Defects

To determine "s," the following equation was used (via SMERFS):

$$\mathrm{MSE}_F = \frac{\sum_{i=s}^{t} [\alpha/\beta(1-\exp(-\beta(i-s+1)))-X_{s,i}]^2}{t-s+1} \tag{2}$$

This equation calculates the MSE for defect counts and cumulative defect counts. "It computes the sum of the squared differences between model predictions and actual cumulative defect counts$X_{s,i}$ in the range $s \le i \le t$." Here, s = 23 was optimal for t = 30.

Figure 14 illustrates the prediction of (1), starting at t = 30 and predicting for $t_2 = 35, 40, 45, 50$, and 55 days, where these represent predicted defects in the intervals 5, 10, 15, 20, and 25 days, respectively, from t = 30. It is seen that the predictions appear good until t_2=55 when the actual defect count takes a sharp turn upward. This is **counter** to what one would expect -- a *decrease* in the *rate* of finding defects as testing continues because the defects become harder to find. When this occurs, it indicates the need for using more of the available data, re-estimating the parameters, and repeating the predictions.

2. Cumulative Defect Count Predictions

As part of a proactive management practice in software reliability, it is advantageous for the manager to be aware of the cumulative count of defects predicted in the software. This information can be used similarly to the defect count predictions but gives a better indicator of the degree of testing problems encountered to date. For this calculation, Equation 3 was used through Statgraphics:

$$\mathbf{F(T)=(\alpha/\beta)[1-exp(-\beta(T-s+1))]+X_{s-1}} \qquad \mathbf{(3)}$$

where X_{s-1} is the defect count in the range 1,s-1.

The results of this calculation did not produce a single curve that accurately matches the true count of cumulative failures. However, upper and lower bound curves were generated as seen in Figure 15.

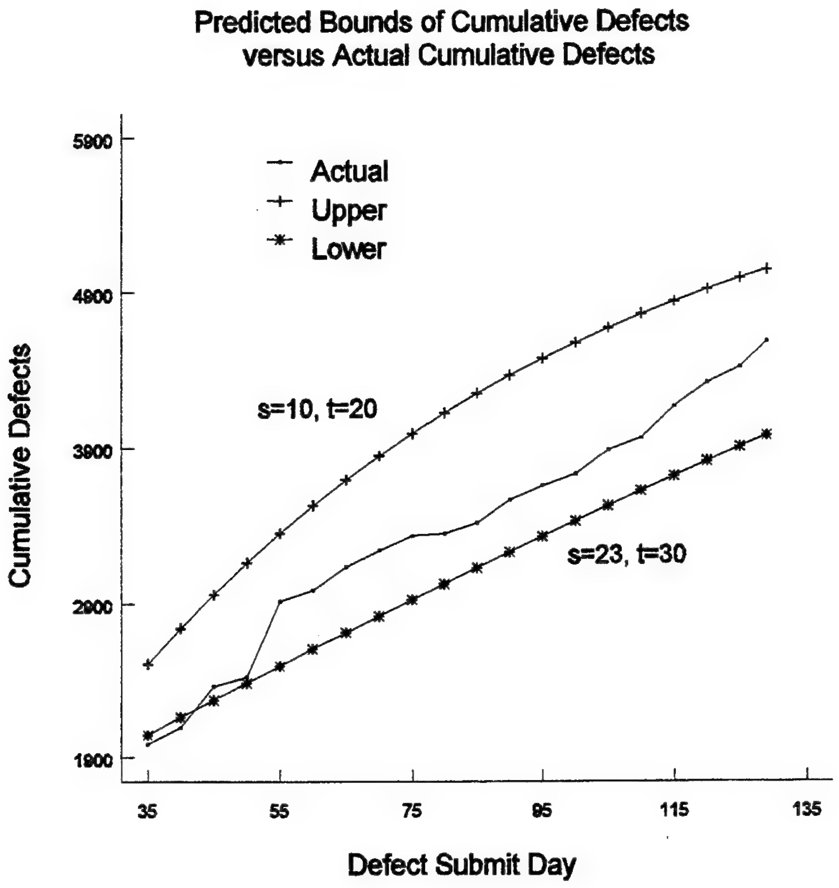


Figure 15. Predicted Bounds of Cumulative Defects vs. Actual Defects

The concept of bounding is important in prediction because the manager would like to know within what limits a quantity is likely to fall. Figure 16 shows that the predicted cumulative defect count for day 129 (May 17, 1995) would fall between 3978 and 5047. The true cumulative defect count for that date is 4584.

3. The Amount of Time Needed to Find the Defects

Each of the previous calculations focuses on the issue that given a specific time interval, as an example, the next two work weeks, how many defects would we predict to occur? A corollary to this question can be proposed. "How much time would it take to find a specific number of defects?" Equation 4, implemented in SMERFS can be used to help answer this question.

$$\mathbf{T_F(t)=[(\log[\alpha/(\alpha-\beta(X_{s,t}+F_t)])/\beta]-(t-s+1)} \quad \mathbf{for} \ (\alpha/\beta)>(X_{s,t}+F_t) \qquad \mathbf{(4)}$$

where (4) is the predicted time (intervals) until the next F_t defects are found, t is the current interval, and $X_{s,t}$ is the cumulative number of defects observed in the range s,t. s = 23 and t=30 were used to produce Figure 16. (The rationale for selecting the proper t and s can be found in Section 2 of this Handbook.) Since the defect count is cumulative, F_t, the time to find the defects increases with time, as can be seen in Figure 16. This is expected since it will take longer to find defects as time passes. The predicted and actual values are comparable until Day 55, when there is an upward increase in predicted time to find the defects. As before when this occurred, there is a need to use more available data, re-estimate the parameters, and repeat the predictions.

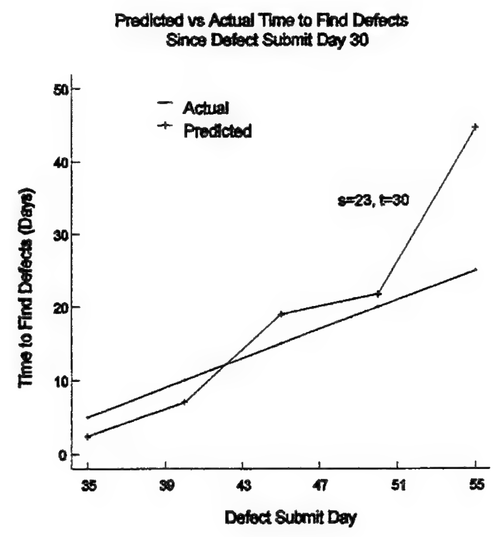


Figure 16. Predicted vs. Actual Time to Find Defects

4. Results

Based on the above predictions, it appears feasible to employ a software reliability model to data obtained for MCTSSA's LOGAIS project. The Schneidewind Software Reliability Model gave predictions comparable to actual defect counts. However, in future calculations, smoothed data would need to be employed to obtain better prediction accuracy.

SECTION 5: MODEL PROPOSAL FOR MCTSSA SYSTEM SURVIVABILITY

This section will present a **proposed** model for use by MCTSSA with its **system** survivability predictions (multi-node configurations). This is in contrast to the previously discussed single node survivability concepts presented in Section 2 of this handbook. Two models take into account different possible system configurations and the impact of server and client failures. These configuration setups can be found in Figures 17 through 19. Of note, this section does not present any actual calculations using MCTSSA test data. It only presents concepts that need to be further evaluated and tested. However, the manager can review this section to better understand the various uses for the Schneidewind Software Reliability Model and its applicability to his organization's system design and configuration.

A. MODEL 1

In this situation there are critical clients: clients with critical functions (e.g., network communication) that must be kept operational for the system to survive. There are also non-critical clients with non-critical functions (e.g., data base query). These clients also act as a backup for the critical clients. The system does not fail unless all the non-critical clients fail and one or more critical clients fail, or one or more servers fail.

1. **Client or server failure**: the application software or operating system in the node ceases to function and the client or server is lost to the distributed system, as a result of a software failure.

2. **System failure**: the system ceases to be operational because either a. all non-critical clients fail *AND* one or more critical clients fail *OR* b. one or more servers fail.

3. **$N_n(t)$**: The number of non-critical clients available in the system at time t, where $N_n(0)$ is the number of non-critical clients at the start of system operation. If a non-critical client fails, the

system can continue to operate -- in a degraded mode -- as long as none of the servers or critical clients fail. In this situation, the function that had been operational on the failed non-critical client can be continued on another client of this type and $N_n(t)$ is decreased by one.

4. **N_c**: The number of critical clients used in the system. If a critical client fails, the system fails, *if there are no non-critical clients available* on which to run the critical client. A change in software configuration may be necessary on the former non-critical client in order to run the critical client. The former non-critical client becomes a critical client, $N_n(t)$ is decreased by one, and the system is run in a degraded mode. As long as the system *remains operational*, N_c is constant.

5. **N_s:** The number of servers used in the system. If a server fails, the system fails.

6. The probability that **all** $N_n(t)$ have failed by time t, given that the software fails, is:

$$\mathbf{P_n(t)=(p_c)^{Nn(t)}}, \qquad \mathbf{(1)}$$

where p_c is the probability that the software failure causes a client failure: p_c = probability (client fails | software fails). Equation (1) assumes that client failures are independent. This is the case because a failure in one client's software would not cause a failure in another client's software. However it is possible that a failure in server software could cause a failure in client software, such as a client accessing a server that has corrupted data. The extent that this could happen depends significantly on whether the client software has been designed to protect against such occurrences. Unless information can be obtained about such occurrences, this factor will be ignored. The probability p_c can be estimated empirically as the ratio of: (client down time caused by software failure)/(scheduled client operating time).

7. The probability that **one or more** N_c fail, given that the software fails, is:

$$\mathbf{P_c=1-(1-p_c)^{Nc}}, \qquad \textbf{(2)}$$

8. The probability that **one or more** N_s fail, given that the software fails, is:

$$\mathbf{P_s=1-(1-p_s)^{Ns}}, \qquad \textbf{(3)}$$

where p_s is the probability that the software failure causes a server failure: p_s = probability (server fails | software fails). Equation (3) assumes that server failures are independent. This is the case because a failure in one server's software would not cause a failure in another server's software. However it is possible that a failure in client software could cause a failure in server software, such as a client with corrupted data accessing a server. The extent that this could happen depends significantly on whether the server software has been designed to protect against such occurrences. Unless information can be obtained about such occurrences, this factor will be ignored. The probability p_s can be estimated empirically as the ratio of: (server down time caused by software failure)/(scheduled server operating time).

9. Combining (1), (2), and (3), the probability of a system failure by time t, given that the software fails, is:

$$\mathbf{P_{sys}(t)=(P_n(t))(P_c)+P_s=[[(p_c)^{Nn(t)}][1-(1-p_c)^{Nc}]]+[1-(1-p_s)^{Ns}]} \qquad \textbf{(4)}$$

The model concepts are illustrated in **Figures 17, 18,** and **19** where there are two servers, five critical clients (C1 ... C5), and five non-critical clients (C6 ... C10). In **Figure 17** one non-critical client, C6, fails; therefore the system survives. In **Figure 18** one of the servers, S1, fails; therefore the system fails. Lastly, in **Figure 19** one of the critical clients, C5, fails and all of the non-critical clients, C6 ... C10, fail; therefore the system fails.

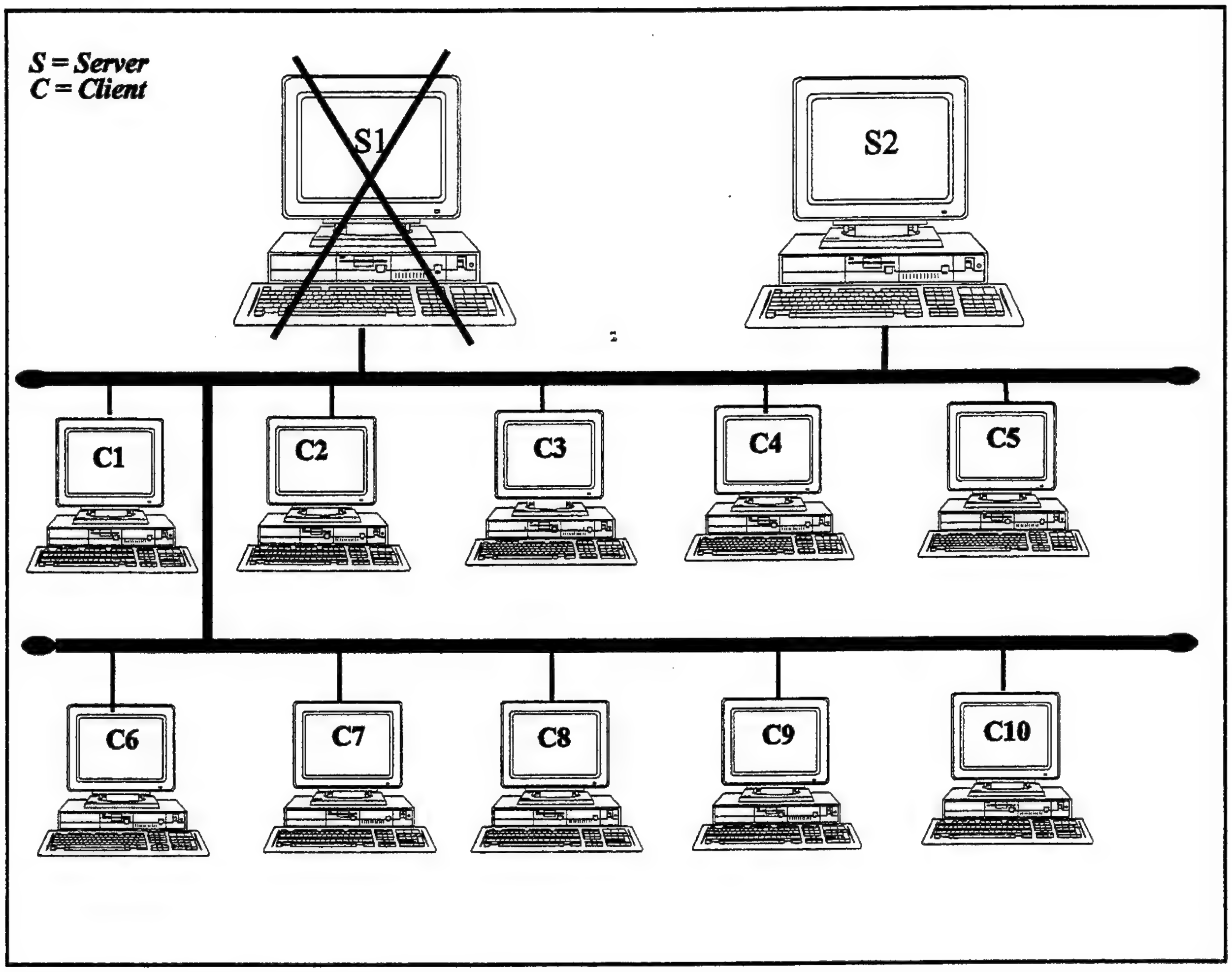


Figure 18. Failing Configuration # 1

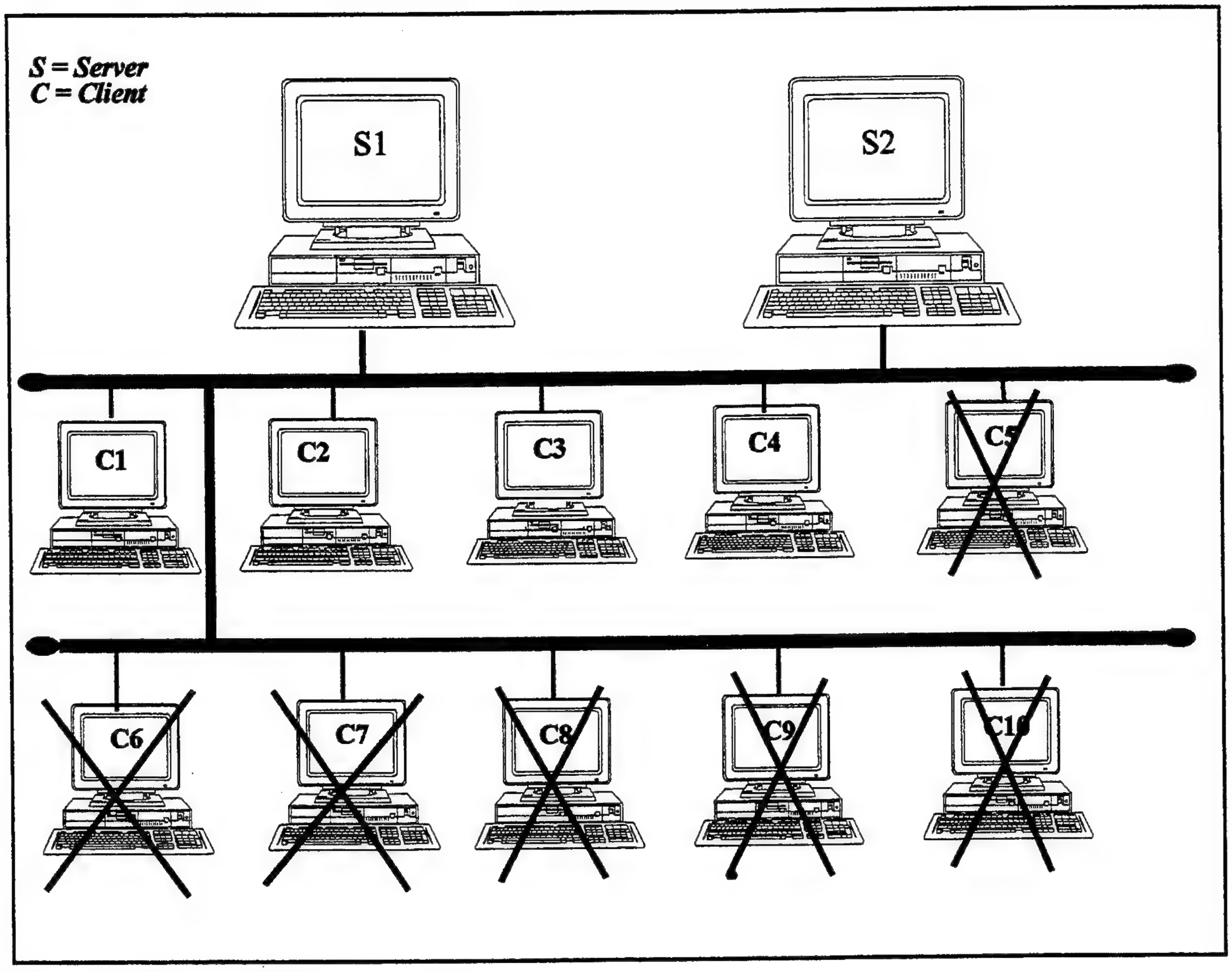


Figure 19. Failing Configuration # 2

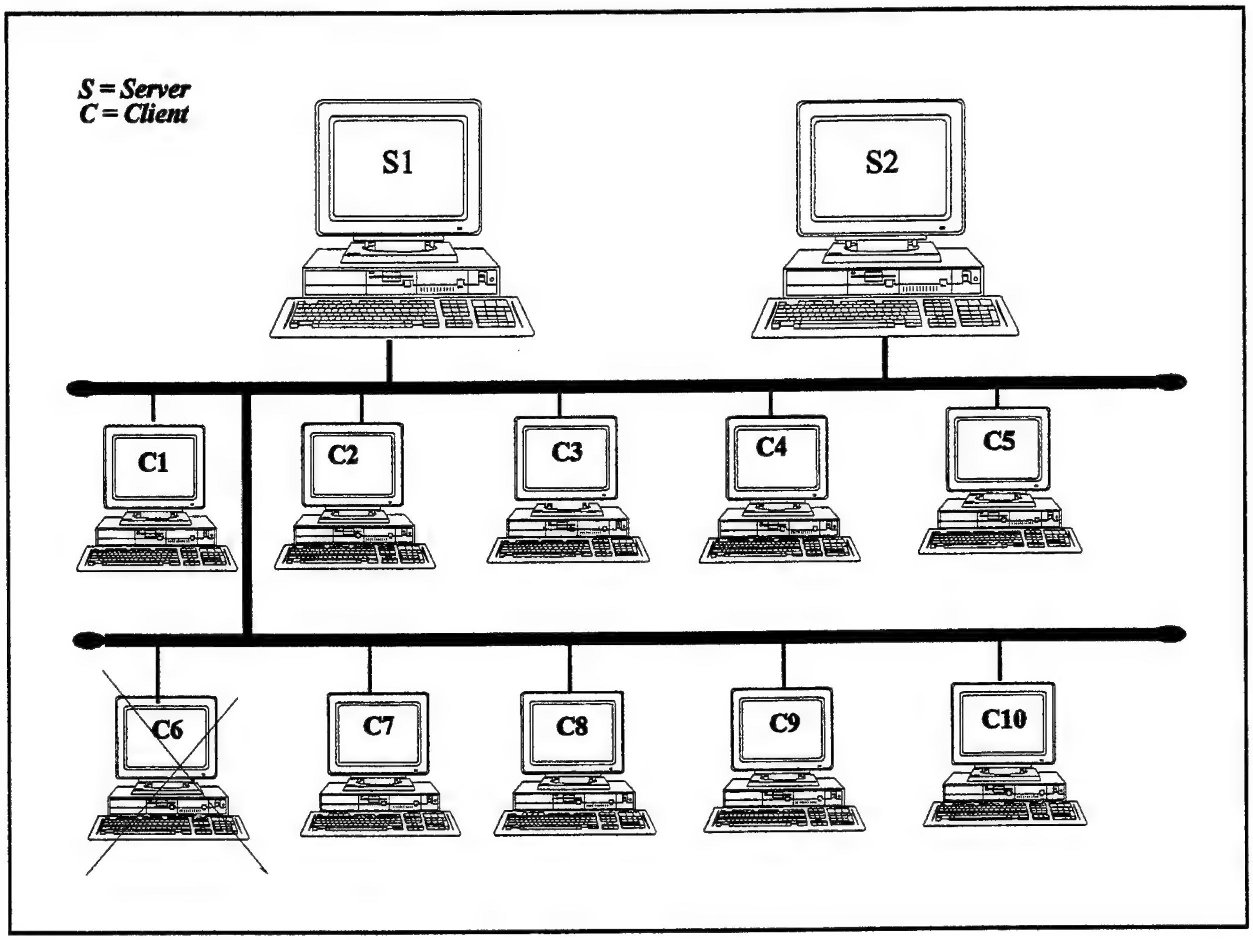


Figure 17. Surviving Configuration

B. MODEL 2

In this situation there are only non-critical clients $N_n(t)$. However there is a minimum number N_{nm} of these clients that must remain operational for the system to survive. Therefore the number that could fail N_f and cause a system failure is $N_f \geq N_n(t)-(N_{nm}-1)$. Thus if there were $N_n(t)=10$ clients at time t and $N_{nm}=3$ clients minimum to keep the system operational, a failure of eight or more clients would reduce the number of clients to less than three.

1. In general the probability of falling below N_{nm} by time t is:

$$\sum_i [N_n(t)!/[(i!)(N_n(t)-i)!]](p_c^i)(1-p_c)^{(Nn(t)-i)} \qquad (5)$$

where $i=N_n(t)-(N_{nm}-1), \ldots, N_n(t)$.

2. Combining (5) and (3), the probability of a system failure by time t, given that the software fails, is:

$$\mathbf{P_{sys}(t)=\sum_i [[N_n(t)!/[(i!)(N_n(t)-i)!]](p_c^i)(1-p_c)^{(Nn(t)-i)}]+[1-(1-p_s)^{Ns}]} \qquad \mathbf{(6)}$$

C. CONCLUSIONS

Based on the above approach, it appears feasible to develop a system software model for distributed systems. The next step is to integrate equations (4) and (6) into the *Schneidewind Software Reliability Model*. Then *MCTSSA* would need to collect system failure data in addition to defect data in order to support model validation. For the purpose of validation it would be necessary to know not only that a defect occurs but, in addition, whether the defect causes a system failure. In addition, information is needed about typical values for p_c and p_s, and an indication of which applications can be represented by Model 1 and which can be represented by Model 2 .

APPENDIX A. SOFTWARE DISCREPANCY REPORT

Date of Failure: Report Originator: Office Code: Project/System Name:	**Discrepancy Report Number**: Telephone Number: Organization: Program Designation: Version:
Category: S (software) H (hardware) P (people)	**Priority/Severity**: 1 (Loss of Life, Mission Aborted) 2 (Degradation in performance) 3 (Operator Annoyance) 4 (Documentation Error) 5 (Error Classification Problem)
Test Procedure: Simulation Used: Linking with: Configuration/Transients in Memory:	
Failure Data: (check one) CPU Time since Last Failure: Clock Time since Last Failure: Manhours expended since Last Failure: **Problem Duplicated**: Yes or No During Run:	**Project Phase**: (check one) Software Requirements Detailed Design Software Integration & Testing Operations / Maintenance Preliminary Design Code / Unit Testing Systems Integration Test
Symptom Classification: **Operating System Crash**: **Program Hang up**: Input Problem: Correct input not accepted Description incorrect or missing Parameters incomplete or missing Output Problem: Wrong format Incorrect result Incomplete or missing output Failed Required Performance: Perceived Total Product Failure: System Error Message: Other (Explain):	

SOFTWARE DISCREPANCY REPORT

Dump Date:
Documents Affected:
Responsible Modules:
Reference Document:
Function Affected:

Project Activity:
Analysis
Inspection
Review
Compile
Audit
Test
Operation
Validation/Qual Test

Actual Cause of Problem
- Product Software/ Database
- Product Hardware
- Test Software
- Documentation
- Interface
- Operator Error
- Enhancement (Perceived inadequacies)

Testing to Verify Fix:
Source of Problem:
Time Required for Analysis:

Disposition:
- Closed:
  - Corrective Action Taken
  - Non-Software Problem
  - Duplicate Problem in STR #
  - Fix not justified
- Open:
  - Deferred to a Later Release
  - Other:
- Merged with another Problem

QA Sign-Off: **Date**:

APPENDIX B. SAMPLE SMERFS PRINT-OUT

```
            SSSSSSS M    M EEEEEEE  RRRRRRR  FFFFFFF  SSSSSSS
                          MMMMMM E      R   R F      S
              SSSSSSS  M  M  M  EEEE     RRRRRRR  FFFF    SSSSSSS
                 S           S M    M E      R  R  F          S
                   SSSSSSS M     M  EEEEEEE R    R F       SSSSSSS

            SOFTWARE REVISION NUMBER FIVE (21 SEPTEMBER 1993)


ENTER OUTPUT FILE NAME FOR THE HISTORY FILE; ZERO IF THE FILE IS
NOT DESIRED, OR ONE FOR DETAILS ON THE FILE.

THE HISTORY FILE IS A COPY OF THE ENTIRE INTERACTIVE SESSION. IT
CAN BE USED FOR LATER ANALYSIS AND/OR DOCUMENTATION.

 a:\smerfs1

ENTER OUTPUT FILE NAME FOR THE  PLOT   FILE; ZERO IF THE FILE IS
NOT DESIRED, OR ONE FOR DETAILS ON THE FILE.

THE PLOT FILE CONTAINS SELECTED DATA AND LABELS TO ALLOW A USER-
SUPPLIED GRAPHICS PROGRAM TO GENERATE HIGH-QUALITY PLOTS.  SINCE
A CHARACTER PLOTTER IS IMPLEMENTED WITHIN THE SMERFS PROGRAM (TO
ENSURE MACHINE PORTABILITY OF THE PACKAGE),  THE USE OF THIS OP-
TION IS HIGHLY RECOMMENDED.

a:\plot

ENTER DESIRED DATA TYPE, OR ZERO FOR A LIST.

     0

THE AVAILABLE DATA TYPES ARE:
 1 WALL CLOCK (WC) TIME-BETWEEN-FAILURES (TBF)
 2 CENTRAL PROCESSING UNITS (CPU) TBF
 3 WC TBF AND CPU TBF
 4 INTERVAL FAULT COUNTS AND TESTING LENGTHS
ENTER DESIRED DATA TYPE.

     4

ENTER ONE FOR A STANDARD SMERFS FILE INPUT; ELSE ZERO.

     1

 ENTER INPUT FILE NAME FOR INTERVAL DATA.

    oi618.in
```

```
THE INPUT OF  18 INTERVAL ELEMENTS WAS PERFORMED.

ENTER INPUT OPTION, OR ZERO FOR A LIST.

    0   (Could enter 3 here.  0 Shows the user a list of options.)

THE AVAILABLE INPUT OPTIONS ARE:
 1 ASCII FILE INPUT
 2 KEYBOARD INPUT
 3 LIST THE CURRENT DATA
 4 RETURN TO THE MAIN PROGRAM
ENTER INPUT OPTION.

    3

 INTERVAL   NO. OF FAULTS      TESTING LENGTH   (These figures are listed in scientific notation.
 ========  ==============     ==============    For example,   .10000000E+01 is the same as 1)
    1     .00000000E+00      .10000000E+01
    2     .00000000E+00      .10000000E+01
    3     .00000000E+00      .10000000E+01
    4     .00000000E+00      .10000000E+01
    5     .30000000E+01      .10000000E+01
    6     .10000000E+01      .10000000E+01
    7     .00000000E+00      .10000000E+01
    8     .10000000E+01      .10000000E+01
    9     .00000000E+00      .10000000E+01
   10      .10000000E+01      .10000000E+01
   11      .10000000E+01      .10000000E+01
   12      .00000000E+00      .10000000E+01
   13      .20000000E+01      .10000000E+01
   14      .00000000E+00      .10000000E+01
   15      .00000000E+00      .10000000E+01
   16      .00000000E+00      .10000000E+01
   17      .00000000E+00      .10000000E+01
   18      .10000000E+01      .10000000E+01

ENTER INPUT OPTION, OR ZERO FOR A LIST.

    0

THE AVAILABLE INPUT OPTIONS ARE:
 1 ASCII FILE INPUT
 2 KEYBOARD INPUT
 3 LIST THE CURRENT DATA
 4 RETURN TO THE MAIN PROGRAM
ENTER INPUT OPTION.

    4
```

```
ENTER ONE FOR THE PROGRAM TO MAKE NEW DATA FILES; ELSE ZERO. THE
RESPONSE WILL BE USED THROUGHOUT THE EXECUTION. A ZERO WILL ALSO
VOID THE DATA RESTORE OPTION IN DATA TRANSFORMATIONS.

    0

ENTER MAIN MODULE OPTION, OR ZERO FOR A LIST.

    0

THE AVAILABLE MAIN MODULE OPTIONS ARE:
 1 DATA INPUT               6 PLOT(S) OF THE RAW DATA
 2 DATA EDIT               7 MODEL APPLICABILITY ANALYSES
 3 UNIT CONVERSIONS         8 EXECUTIONS OF THE MODELS
 4 DATA TRANSFORMATIONS 9 STOP EXECUTION OF SMERFS
 5 DATA STATISTICS

ENTER MAIN MODULE OPTION.
    8

ENTER COUNT MODEL OPTION, OR ZERO FOR A LIST.

    0

THE AVAILABLE FAULT COUNT MODELS ARE:
 1 THE BROOKS AND MOTLEY MODEL
 2 THE GENERALIZED POISSON MODEL
 3 THE NON-HOMOGENEOUS POISSON MODEL
 4 THE SCHNEIDEWIND MODEL
 5 THE S-SHAPED RELIABILITY GROWTH MODEL
 6 RETURN TO THE MAIN PROGRAM
ENTER MODEL OPTION.

    4

ENTER ONE FOR SCHNEIDEWIND MODEL DESCRIPTION; ELSE ZERO.

    0
ENTER ONE FOR DESCRIPTION OF TREATMENT TYPES; ELSE ZERO.

    0

ENTER ONE TO INVESTIGATE FOR THE OPTIMUM S (USING TREATMENT TYPE
NUMBER 2); ELSE ZERO TO CONTINUE WITH THE MODEL EXECUTION.

    1

 ENTER RANGE OVER WHICH S  SHOULD BE TESTED.  NOTE, AN EXECUTION
 ON A GIVEN S WHICH FAILED THE CONVERGENCE CRITERIA  WILL NOT BE
```

```
INCLUDED IN THE FOLLOWING RESULTS TABLE.  THE OPTIMUM S FOR EI-
THER MSE-F OR MSE-T IS THE ONE  RESULTING IN THE SMALLEST VALUE
FOR YOUR CHOSEN CRITERIA.
     1        9       (Enter as 1,9)

 S   BETA       ALPHA       WLS       MSE-F      MSE-T
=== ========== ========== ========== ========== ==========
 1 .37154E-02 .57434E+00 .71189E+00 .89573E+00 .15098E+01
 2 .25076E-01 .72250E+00 .84899E+00 .68418E+00 .12947E+01
 3 .52370E-01 .92300E+00 .10130E+01 .47735E+00 .10803E+01
 4 .88195E-01 .12021E+01 .12214E+01 .34612E+00 .86076E+00
 5 .13700E+00 .16059E+01 .15409E+01 .47758E+00 .60788E+00  (Record these values for later
 6 .51401E-01 .73825E+00 .58125E+00 .24450E+00 .11042E+01   calculations.  They are the  minimum
 7 .28025E-01 .58878E+00 .50090E+00 .30476E+00 .13863E+01   MSE values for their columns).
 9 .60985E-01 .66786E+00 .61535E+00 .28068E+00 .13683E+01

ENTER ONE TO INVESTIGATE ANOTHER RANGE FOR S; ELSE ZERO.

     1

ENTER RANGE OVER WHICH S  SHOULD BE TESTED.  NOTE, AN EXECUTION
ON A GIVEN S WHICH FAILED THE CONVERGENCE CRITERIA  WILL NOT BE
INCLUDED IN THE FOLLOWING RESULTS TABLE.  THE OPTIMUM S FOR EI-
THER MSE-F OR MSE-T IS THE ONE  RESULTING IN THE SMALLEST VALUE
FOR YOUR CHOSEN CRITERIA.

     1        10      (This is an optional comparison).

 S   BETA       ALPHA       WLS       MSE-F      MSE-T
=== ========== ========== ========== ========== ==========
 1 .37154E-02 .57434E+00 .71189E+00 .89573E+00 .15098E+01
 2 .25076E-01 .72250E+00 .84899E+00 .68418E+00 .12947E+01
 3 .52370E-01 .92300E+00 .10130E+01 .47735E+00 .10803E+01
 4 .88195E-01 .12021E+01 .12214E+01 .34612E+00 .86076E+00
 5 .13700E+00 .16059E+01 .15409E+01 .47758E+00 .60788E+00
 6 .51401E-01 .73825E+00 .58125E+00 .24450E+00 .11042E+01
 7 .28025E-01 .58878E+00 .50090E+00 .30476E+00 .13863E+01
 9 .60985E-01 .66786E+00 .61535E+00 .28068E+00 .13683E+01

ENTER ONE TO INVESTIGATE ANOTHER RANGE FOR S; ELSE ZERO.

     0

ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL
EXECUTION.

     2

ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).

     5   (This value corresponds to the minimum MSE-T).
```

```
TREATMENT 2 MODEL ESTIMATES ARE:
 BETA                .13700E+00
 ALPHA               .16059E+01
 TOTAL NUMBER OF FAULTS    .11722E+02        (These are the key values for the calculation.)
  PLUS THOSE SKIPPED     .00000E+00 IN PERIODS 1 THROUGH  4
 # OF FAULTS REMAINING    .17221E+01
 WEIGHTED SUMS-OF-SQUARES
  BETWEEN PREDICTED AND
  OBSERVED FAULTS         .15409E+01
 MEAN SQUARE ERROR FOR
  CUMULATIVE FAULTS       .47758E+00
 MEAN SQUARE ERROR FOR
  TIME TO NEXT FAILURE    .60788E+00

THE AVAILABLE FUTURE PREDICTIONS ARE:
 1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
 2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

    2

 ENTER VALUE OF M (BETWEEN ONE AND  .17221E+01), OR ZERO TO END.

   1.000000000000000

 # OF PERIODS EXPECTED     .63443E+01  (This shows the predicted time to next failure).


  ENTER VALUE OF M (BETWEEN ONE AND  .17221E+01), OR ZERO TO END.

   1.722000000000000

 # OF PERIODS EXPECTED     .71759E+02  (This shows the predicted amount of time it would take
                                        to reduce the number of remaining failures to .0001).

 ENTER VALUE OF M (BETWEEN ONE AND  .17221E+01), OR ZERO TO END.

0.000000000000000E+000

THE AVAILABLE FUTURE PREDICTIONS ARE:
 1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
 2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

    0

ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL
EXECUTION.

    2
```

```
ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).

    6      (This corresponds to the minimum MSE-F value).

TREATMENT 2 MODEL ESTIMATES ARE:
BETA                  .51401E-01
ALPHA                 .73825E+00
TOTAL NUMBER OF FAULTS     .14363E+02      (These are the key values of interest).
 PLUS THOSE SKIPPED      .30000E+01 IN PERIODS 1 THROUGH   5
# OF FAULTS REMAINING     .43626E+01
WEIGHTED SUMS-OF-SQUARES
 BETWEEN PREDICTED AND
 OBSERVED FAULTS          .58125E+00
MEAN SQUARE ERROR FOR
 CUMULATIVE FAULTS        .24450E+00
MEAN SQUARE ERROR FOR
 TIME TO NEXT FAILURE     .11042E+01

THE AVAILABLE FUTURE PREDICTIONS ARE:
 1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
 2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

    1

ENTER NUMBER OF PERIODS TO EXAMINE, OR ZERO TO END.

   1.000000000000000     (Enter as 1)

# OF FAULTS EXPECTED      .36888E+00    (This predicts the remaining number of faults in one more
                                         period).

ENTER NUMBER OF PERIODS TO EXAMINE, OR ZERO TO END.

0.000000000000000E+000    (Enter as 0)

THE AVAILABLE FUTURE PREDICTIONS ARE:
 1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
 2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

    0

ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL
EXECUTION.

    4
```

```
ENTER ONE TO PERFORM AN ANALYSIS OF THE MODEL FIT USING THE PRE-
DICTIONS OF THIS MODEL; ELSE ZERO.

     0



ENTER COUNT MODEL OPTION, OR ZERO FOR A LIST.

     0

THE AVAILABLE FAULT COUNT MODELS ARE:
 1 THE BROOKS AND MOTLEY MODEL
 2 THE GENERALIZED POISSON MODEL
 3 THE NON-HOMOGENEOUS POISSON MODEL
 4 THE SCHNEIDEWIND MODEL
 5 THE S-SHAPED RELIABILITY GROWTH MODEL
 6 RETURN TO THE MAIN PROGRAM

ENTER MODEL OPTION.

     6

ENTER MAIN MODULE OPTION, OR ZERO FOR A LIST.

     0

THE AVAILABLE MAIN MODULE OPTIONS ARE:
 1 DATA INPUT                    6 PLOT(S) OF THE RAW DATA
 2 DATA EDIT                     7 MODEL APPLICABILITY ANALYSES
 3 UNIT CONVERSIONS              8 EXECUTIONS OF THE MODELS
 4 DATA TRANSFORMATIONS          9 STOP EXECUTION OF SMERFS
 5 DATA STATISTICS
ENTER MAIN MODULE OPTION.

     9


THE SMERFS EXECUTION HAS ENDED.
```

APPENDIX C. LOGAIS CHRONOLOGICAL DEFECT COUNTS

Table 3. *LOGAIS* Chronological Defect Counts

Count Interval (*t*)	Defect ID Range	Number of Defects	Summit Date	Day
1	1-120	120	11/11/94	Fri
2	121-305	185	11/12/94	Sat
3	306	1	11/13/94	Sun
4	307-497	191	11/14/94	Mon
5	498-710	213	11/15/94	Tue
	5 Day Total	**710**		
	Cumulative	**710**		
6	711-888	178	11/16/94	Wed
7	889-942	54	11/17/94	Thu
8	943-981	39	11/18/94	Fri
9	982-996	15	11/19/94	Sat
10	997-1024	28	11/20/94	Sun
	5 Day Total	**314**		
	Cumulative	**1024**		
11	1025-1123	99	11/21/94	Mon
12	1124-1193	70	11/22/94	Tue
13	1194-1253	60	11/23/94	**Wed**
14	1254-1263	10	11/25/94	**Fri**
15	1264-1368	105	11/28/94	**Mon**
	5 Day Total	**344**		
	Cumulative	**1368**		
16	1369-1483	115	11/29/94	Tue
17	1484-1565	82	11/30/94	Wed
18	1566-1624	59	12/1/94	Thu

19	1625-1697	73	12/2/94	Fri
20	1698-1703	6	12/3/94	Sat
	5 Day Total	**335**		
	Cumulative	**1703**		
21	1704-1721	18	12/4/94	Sun
22	1722-1740	19	12/5/94	Mon
23	1741-1772	32	12/6/94	Tue
24	1773-1803	31	12/7/94	Wed
25	1804-1823	20	12/8/94	Thu
	5 Day Total	**120**		
	Cumulative	**1823**		
26	1824-1830	7	**12/9/94**	**Fri**
27	1831-1840	10	**12/15/94**	**Thu**
28	1841-1861	21	**12/19/94**	**Mon**
29	1862-1915	54	12/20/94	Tue
30	1916-1929	14	12/21/94	Wed
	5 Day Total	**106**		
	Cumulative	**1929**		
31	1930-1935	6	12/22/94	Thu
32	1936-1960	25	**12/23/94**	**Fri**
33	1961-1964	4	**12/28/94**	**Wed**
34	1965-1982	18	12/29/94	Thu
35	1983-1985	3	**12/30/94**	**Fri**
	5 Day Total	**56**		
	Cumulative	**1985**		
36	1986	1	**1/3/95**	**Tue**
37	1987-2000	14	1/4/95	Wed
38	2001-2003	3	1/5/95	Thu
39	2004-2027	24	**1/6/95**	**Fri**
40	2028-2093	66	1/9/95	Mon

	5 Day Total	**108**		
	Cumulative	**2093**		
41	2094-2157	64	1/10/95	Tue
42	2158-2231	74	1/11/95	Wed
43	2232-2292	61	1/12/95	Thu
44	2293-2358	66	1/13/95	Fri
45	2359-2362	4	**1/14/95**	**Sat**
	5 Day Total	**269**		
	Cumulative	**2362**		
46	2363-2372	10	**1/16/95**	**Mon**
47	2373-2390	18	1/17/95	Tue
48	2391-2399	9	1/18/95	Wed
49	2400-2405	6	1/19/95	Thu
50	2406-2424	19	**1/20/95**	**Fri**
	5 Day Total	**62**		
	Cumulative	**2424**		
51	2425-***	48	**1/24/95**	**Tue**
52	2426-***	44	1/25/95	Wed
53	2430-***	145	1/26/95	Thu
54	2433-***	227	**1/27/95**	**Fri**
55	2446-2473	28	**1/30/95**	**Mon**
	5 Day Total	**492**		
	Cumulative	**2916**		
56	2474-2480	7	1/31/95	Tue
57	2481-2486	6	2/1/95	Wed
58	2487-2510	24	2/2/95	Thu
59	2511-2529	19	**2/3/95**	**Fri**
60	2530-2543	14	2/6/95	Mon
	5 Day Total	**70**		
	Cumulative	**2986**		

61	2544***	53	2/7/95	Tue
62	3040-3067	28	2/8/95	Wed
63	3068-3099	32	2/9/95	Thu
64	3100-3110	11	**2/10/95**	**Fri**
65	3111-3137	27	2/13/95	Mon
	5 Day Total	**151**		
	Cumulative	**3137**		
66	3138-3146	9	2/14/95	Tue
67	3147-3167	21	2/15/95	Wed
68	3168-3213	46	2/16/95	Thu
69	3214-3233	20	**2/17/95**	**Fri**
70	3234-3242	9	**2/21/95**	**Tue**
	5 Day Total	**105**		
	Cumulative	**3242**		
71	3243-3260	18	2/22/95	Wed
72	3261-3314	54	2/23/95	Thu
73	3315-3320	6	**2/24/95**	**Fri**
74	3321-3324	4	**2/27/95**	**Mon**
75	3325-3334	10	2/28/95	Tue
	5 Day Total	**92**		
	Cumulative	**3334**		
76	3335-3340	6	3/1/95	Wed
77	3341	1	3/2/95	Thu
78	3342-3343	2	**3/3/95**	**Fri**
79	3344-3347	4	**3/6/95**	**Mon**
80	3348-3349	2	3/7/95	Tue
	5 Day Total	**15**		
	Cumulative	**3349**		
81	3350-3362	13	**3/8/95**	**Wed**
82	3363-3368	6	**3/10/95**	**Fri**
83	3369-3379	11	**3/13/95**	**Mon**

84	3380-3383	4	3/14/95	Tue
85	3384-3419	36	3/15/95	Wed
	5 Day Total	**70**		
	Cumulative	**3419**		
86	3420-3431	12	3/16/95	Thu
87	3432-3447	16	**3/17/95**	**Fri**
88	3448-3492	45	**3/20/95**	**Mon**
89	3493-3530	38	3/21/95	Tue
90	3531-3566	36	3/22/95	Wed
	5 Day Total	**147**		
	Cumulative	**3566**		
91	3567-3601	35	3/23/95	Thu
92	3602-3616	15	**3/24/95**	**Fri**
93	3617-3635	19	**3/27/95**	**Mon**
94	3636-3652	17	3/28/95	Tue
95	3653-3658	6	3/29/95	Wed
	5 Day Total	**92**		
	Cumulative	**3658**		
96	3659-3681	23	3/30/95	Thu
97	3682-3693	12	**3/31/95**	**Fri**
98	3694-3710	17	**4/3/95**	**Mon**
99	3711-3726	16	4/4/95	Tue
100	3727-3731	5	4/5/95	Wed
	5 Day Total	**73**		
	Cumulative	**3731**		
101	3732-3769	38	4/6/95	Thu
102	3770-3840	71	**4/7/95**	**Fri**
103	3841	1	**4/10/95**	**Mon**
104	3842-3856	15	4/11/95	Tue
105	3857-3885	29	4/12/95	Wed
	5 Day Total	**154**		

	Cumulative	**3885**		
106	3906-***	23	4/13/95	Thu
107	3909-3923	15	**4/14/95**	**Fri**
108	3924-3932	9	**4/16/95**	**Sun**
109	3933-3949	17	4/17/95	Mon
110	3950-3963	14	4/18/95	Tue
	5 Day Total	**78**		
	Cumulative	**3963**		
111	3964-4033	70	4/19/95	Wed
112	4034-4079	46	**4/20/95**	**Thu**
113	4080-4107	28	**4/25/95**	**Tue**
114	4108-4132	25	4/26/95	Wed
115	4133-4167	35	4/27/95	Thu
	5 Day Total	**204**		
	Cumulative	**4167**		
116	4168-4176	9	**4/28/95**	**Fri**
117	4177-4185	9	**5/1/95**	**Mon**
118	4186-4207	22	5/2/95	Tue
119	4208-4287	80	5/3/95	Wed
120	4288-4320	33	5/4/95	Thu
	5 Day Total	**153**		
	Cumulative	**4320**		
121	4321-4328	8	**5/5/95**	**Fri**
122	4329-4343	15	**5/8/95**	**Mon**
123	4344-4356	13	5/9/95	Tue
124	4357-4367	11	5/10/95	Wed
125	4368-4418	51	5/11/95	Thu
	5 Day Total	**98**		
	Cumulative	**4418**		
126	4419-4504	86	**5/12/95**	**Fri**
127	4505-4513	9	**5/15/95**	**Mon**
128	4514-4555	42	5/16/95	Tue

129	4556-4584	29	5/17/95	Wed
	TOTAL	**4584**		

*** Indicates discontinuous sequences of IDs for Submit Date. First ID of first sequence shown.
Bolding indicates breaks in defect submit dates.

LIST OF REFERENCES

[AIA93] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[BIL94] C. Billings, et al, "Journey to a Mature Software Process", IBM Systems Journal Vol. 33, No. 1, 1994, pp. 46-61.

[FAR93] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

[IEE93] ANSI/IEEE Standard for a Software Quality Metrics Methodology, IEEE 1061 June, 1993.

[KEL95] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX, March 28, 1995, pp. 1-8.

[MUS87] John Musa, et al, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

[SCH93] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.

[SCH92a] Norman F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, May 1992, pp. 410-422.

[SCH92] Norman F. Schneidewind and T.W. Keller, "Application of Reliability Models to the Space Shuttle", IEEE Software, Vol. 9, No. 4, July 1992 pp. 28-33.

[SCH75] Norman F. Schneidewind, "Analysis of Error Processes in Computer Software", Proceedings of the International Conference on Reliable Software, IEEE Computer Society, 21-23 April 1975, pp. 337-346.

[WAR94] Kenneth M. Warburton, "Towards Better Quality and Reliability in the Software Reuse Library Environment," Master's Thesis, Naval Postgraduate School, March 1994.

APPENDIX B. MCTSSA SRE Training Plan

MCTSSA SOFTWARE RELIABILITY ENGINEERING TRAINING PLAN

Final Version: January 10, 1996

Dr. Norman F. Schneidewind
LCDR Judie A. Heineman

Naval Postgraduate School
Code SM/Ss
Monterey, California 93943

Voice: (408) 656-2719
Fax: (408) 656-3407

Internet: schneidewind@nps.navy.mil

MCTSSA TRAINING PLAN
January 10, 1996

I. TOPIC ONE: **Defining a Software Reliability Engineering Program**

A. Project Background and Purpose

i. When an organization contracts for new software, it expects to receive a "quality" product. But how does it know that what it is receiving will perform as expected? What data was collected and what tests were run to show that the software is "good?" How should the manager interpret the test results; what reliability information do they show? What information should ***the manager*** gather to make a decision on how reliable the software is? This training plan is designed to provide the training necessary for Marine Corps personnel to implement and manage the software reliability engineering program which is described in the companion document: **MCTSSA SOFTWARE RELIABILITY HANDBOOK, January 10, 1996.** Where appropriate, sections in this training plan are cross referenced to the applicable sections and pages in the handbook. Note: The handbook covers many types of software reliability predictions. Only selected ones are covered in this training plan to illustrate the prediction process.

ii. Quality is one of the user-oriented characteristics of software that is most challenging to define quantitatively. To some, quality can be indirectly measured by reliability. With this in mind, *software reliability* means that the software will perform as expected for a specific period of time before it malfunctions. Because reliability relates to the operation of software, it is appropriately related to the term quality.

iii. The Naval Postgraduate School in Monterey, CA was tasked with designing a Software Reliability Engineering (SRE) Program for MCTSSA for use in its Automated Information Systems. This SRE Program focuses on implementing certain managerial procedures which allow for data collection and analysis -- the cornerstones of the program. These procedures are based on an understanding of the definition of reliability, its significance in determining the quality of a product, and its usefulness in making managerial decisions. The program's ultimate objective is to **PREDICT** software reliability using software failure data collected during the program's development and design phases. This data is then input into the Schneidewind Software Reliability Model which will provide the user with information about the software's **PREDICTED** reliability. Specific uses for this predicted reliability will be discussed further in the following presentations.

B. Anticipated Challenges

i. When working with data, the user must anticipate the challenges he will face in collecting the proper data and subsequently manipulating the data into a format usable for his purposes. Frequently, raw data is unusable in its native form, and must be "smoothed" into a format that can be readily applied to the model. Some typical "smoothing" techniques include: averaging and moving average. Detailed discussion of these techniques is beyond the scope of this training session.

C. SRE Definition and Managerial Application

i. Software Reliability Engineering (SRE) is a new discipline that is maturing as more organizations see the need to develop standard reliability practices. The American Institute of Aeronautics and Astronautics (AIAA) Recommended Practice on Software Reliability defines SRE as "the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems."

ii. Handling, identifying and correcting faults are significant concerns for the manager because the entire software reliability process is expensive. "It also impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time and peripherals requirements)." (AIAA) This addresses the key issue regarding SRE -- **it provides the manager with information about which he can make informed decisions**. There will always be a tradeoff between reliability, seen as the failure rate, and cost. (Cost is directly related to testing time). The manager will need to decide on a certain level of reliability for the product resulting in a set cost. Higher reliability will result in higher cost. The converse is also true. This is seen in Figure 1 (in the handbook, page 4).

D. Faults vs. Failures

i. As with any intellectual product, errors in design may occur. An error can be defined as **"a discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition."** (AIAA) In software, these errors may appear while completing requirements formulation or, as is often the case, during design, coding, and testing the product.

ii. The software development process should include measures to discover and correct faults resulting from these errors. [In this context, **faults are defined as "defects in the code that can be the cause of one or more failures."**] (AIAA) These measures can address reviews, audits, screening by language-dependent tools, and several layers of testing. One way to reduce the number and criticality of errors is by modeling the effects of the remaining faults in the delivered product. This can be achieved through a dedicated measurement process by which each defect or fault is noted and formally recorded for inclusion in the reliability model.

iii. As a point of clarification, a **fault is technically different from a failure.** A failure can be defined as "the inability of a system or system component to perform a required function within specified limits" or the "departure of program operation from program requirements." (AIAA) In simpler terms, *a fault usually leads to a failure.*

E. Components of an SRE Program

i. A model is chosen for implementation in the SRE program. It should have the ability to make the types of predictions desired by the user, with a specified accuracy. In MCTSSA's SRE Program, the Schneidewind Software Reliability Model, one of the four models recommended by the "American Standards Institute/American Institute of Aeronautics and Astronautics Recommended Practice on Software Reliability", is used. This model can be used to predict the following: Time to Next Failure, Cumulative Failures for a Specified Time, Remaining Failures and Fraction of Remaining Failures, Total Failures over the Life of the Software, Test Time to Achieve Specified Remaining Failures, and Operational Quality.

ii. A successful software reliability program does not consist of just a model; it also consists of the support structure and various definitions:

* **reliability requirements;**
* **reliability measurements to meet those requirements;**
* **data collection procedures to obtain the necessary data;**
* **severity levels of failures;**
* **applications of reliability predictions;**
* **interpretation of model predictions; and**
* **user feedback for model improvements.**

Although the conceptualization of the model does not occur in a sequence of steps, its *implementation* does. The practitioner can best understand this process from a description of the chronology of implementing and applying the model. Therefore, this approach will be used in explaining the process. To illustrate the process, many equations, figures, and tables will be used. Many real-world, actually *out-of this-world,* examples from the *Space Shuttle* will be used, because the process can be illustrated with real data, and real predictions. However, it should not be concluded that the examples are not applicable to MCTSSA; they are. The approach is generic and its feasibility can be tested against MCTSSA systems. The *Shuttle* is a safety critical system where human life and expensive equipment are at risk. This is also the case with MCTSSA systems.

II. **TOPIC TWO: Implementing a Software Reliability Program**

A. Without management's involvement in the SRE process, all efforts put forth to produce a valid SRE program would be in vain. Management must provide resources to establish a working, practical program. These resources include personnel, time, and information resources such as computers and software to perform the calculations. Once this is established, an SRE program can begin.

B. The SRE process normally consists of two phases: stating the organization's reliability goals and testing.

i. The organization's *reliability goals* can be ideal or conceptual but must have some basis in reality. A goal of "0%" defects might be the ideal, but this has a slim likelihood of occurring in the real world. If it could occur, it will cost an extraordinarily large sum of money to obtain.

ii. The second phase of the SRE involves *testing*. It is here that the failure data is collected and formatted for inclusion in the model of choice. **It is the data that allows the predictions for reliability to be made.** The test plan used must be consistent with the established goals. If a goal is to have the predicted maximum number of remaining failures less than one, then the test plan must be able to predict the remaining number of failures in the software. The tests provide insight into the future -- what may occur as a result of using this software that has been tested. This insight is used to either forge ahead with actual implementation of the software or return to the drawing board and reassess the system. It will provide an indication as to whether or not additional testing is needed because the results to date may be inconclusive or show an undesirable trend. The test results also allow the manager to prioritize his assets. It can help him to decide where he should assign his resources. Is Module C predicted to be more reliable than Module B? If this is true, he may decide to allocate the majority of his resources to Module B to improve its reliability.

C. The following steps should be considered by any organization as it begins to develop a software reliability program. Each step is discussed in the handbook, pages 6-14. The recommended steps in implementing an SRE include the following:

- Establish a Measurement Framework
- Collect the Data
- Establish Problem Severity Levels
- Estimate Model Parameters
- Select the Optimal Set of Failure Data
- Identify the Operational Profile
- Make Reliability Predictions
- Validate the Model
- Make Reliability Decisions
- Use Software Reliability Tools

D. SRE Implementation Phases

i. **Step 1**: State the Reliability Requirement. In this step, the software manager should describe the conditions that must be fulfilled for the software to be considered satisfactory (reliable). An example of such a requirement may be "No software failure that would result in loss of life, loss of mission, or abort of mission."

ii. **Step 2**: Establish a Measurement Framework. The organization should consider a comprehensive measurement plan that would include *indirect* measures of quality like problem report counts, size and complexity metrics. Figure 2 (in the handbook, page 9) captures this idea. In this diagram, *Level 1* shows the most direct measurement (e.g., a *time between failures*); *Level 2* shows an indirect measurement (e.g., *discrepancy report count*) one level removed from the direct measurement; and *Level 3* shows an indirect measurement two levels removed from the direct measurement (e.g., *size* and *complexity*). The advantage of *Level 1* measurements is that they are the most accurate representations of reliability; their disadvantage is that they cannot be collected until the software is tested. Conversely, the indirect measurements are less accurate as representations of reliability, but they can be collected earlier in the development process.

iii. **Step 3:** Collect the Data. Data could be collected in the format as shown in Table 1(in the handbook, page 10). For each system, there should be a brief description of its purpose and functions. The ***Days*** # field could be noted in *hours* or *minutes*, as appropriate. It is recommended that the ***Problem Report ID*** field be coded to indicate Software (S) failure, Hardware (H) failure, or People (P) failure. A more detailed tracking report can be implemented as the organization becomes more advanced in its SRE process. An example of such a report is found in the handbook in Appendix A.

iv. **Step 4**: Establish Problem Severity Levels. The levels assigned in this step are purely discretionary and must be determined by the command. Some practical, recommended severity level descriptions are as follows.

Level 1. Loss of life, loss of mission, abort mission
Level 2. Degradation in performance
Level 3. Operator annoyance
Level 4. System OK, but documentation in error
Level 5. Error in classifying a problem (i.e., no problem existed)

Note: Not all problems (faults) result in failures.

v. **Step 5**: Estimate Model Parameters. Once a model has been chosen to be applicable to a particular system, the necessary model parameters must be estimated, using SMERFS . Three parameters are used in the Schneidewind Model and will be used for MCTSSA: α , which is the failure rate at the beginning of interval "s," β , which is the failure rate per failure, and "s," the first interval used in parameter estimation. These parameters will be discussed later in the presentation and are only presented here as an introduction to the terminology used in the methodology.

vi. **Step 6**: Select the Optimal Set of Failure Data. This stage selects the subset of failure data, starting with interval, "s" through "t," the last observed interval. The objective here is to find the set of failure data that will give the best parameter estimates and the most accurate predictions.

vii. **Step 7**: Identify the Operational Profile. The operational profile describes the system's environment. It includes the input variables (e.g. a listing of available equipment or a ship's destination), the functional environment of the program (i.e. a specific function the system is to perform such as sorting the available equipment by minor property number), and the output variable (e.g. a printout of the ship's destinations for the next two months). In this framework, a failure can be seen as a departure of the output variable from what it is expected to be.

viii. **Step 8**: Make Reliability Predictions. This step is the key to predicting the reliability of the AIS. Each of the listed predictions is described in detail in the Basic Concepts section (in the handbook, starting on page 17). The possible predictions resulting from the model application are:

a. Time to Next Failure
b. Cumulative Failures for a Specified Time
c. Remaining Failures and Fraction of Remaining Failures
d. Total Failures over the Life of the Software
e. Test Time to Achieve Specified Remaining Failures
f. Operational Quality

ix. **Step 9**: Validate the Model. This step evaluates the model to determine if it measures what the model is designed to measure. The predicted values are compared to the actual values. Once the two values are compared with each other, a determination of the model's validity can be made. As an example, if the model predicts the time to next failure will be two periods, the predicted time would be compared to the actual time. This step is accomplished after certain numbers and types of predictions have been made. If, however, the values do not compare favorably, the data used in the model should be carefully examined to identify if anything unusual can be found. If the data appears valid, and the model prediction does not match reality, different models would need to be investigated. For the purposes of this handbook, the Schneidewind Reliability Model will be used.

x. **Step 10**: Make Reliability Decisions. The purpose of implementing a reliability program is to provide the manager with additional information through which he can make informed decisions. Reliability decisions such as "Is the software safe enough to use such that it will not cause or result in loss of life?" can be made as a result of the model's predictions. For this example, the predicted remaining failures must be less than a specified critical value and the predicted time to next failure must be at least as large at the mission execution time plus some safety margin. This example will be addressed later in the handbook using numerical examples.

xi. **Step 11**: Use Software Reliability Tools. There are software reliability tools available to make the model calculations easier to achieve. The *Statistical Modeling and Estimation of Reliability Functions for Software*, SMERFS, is a software package available for this purpose. A sample SMERFS session is outlined in the Testing Procedures section of the handbook, page 41. A complete discussion of SMERFS will be presented towards the end of this training session. Additionally, *Statgraphics* is a software tool available that provides additional prediction equations of the Schneidewind Software Reliability model that are not included in SMERFS. This tool will be further discussed in the tutorial at the end of the training session.

E. SRE Implementation Summary

i. The organization's *reliability goals* can be ideal or conceptual but must have some basis in reality. A goal of "0%" defects might be the ideal, but this has a slim likelihood of occurring in the real world. If it could occur, it will cost an extraordinarily large sum of money to obtain.

ii. The second phase of the SRE involves testing. It is here that the failure data is collected and formatted for inclusion in the model of choice. The test plan used must be consistent with the goals established. If a goal is to have a maximum number of remaining failures of less than one, then the test plan must be able to predict the remaining number of failures in the software. The tests provide insight into the future - - what may occur as a result of using this software that has been tested. This insight is used to either forge ahead with actual implementation of the software or return to the drawing board and reassess the system. It will provide an indication as to whether or not additional testing is needed because the results to date may be inconclusive or show an undesirable trend. The test results also allow the manager to prioritize his assets. It can help him to decide where he should assign his resources. Is Module C predicted to be more reliable than Module B? If this is true, he may decide to allocate the majority of his resources to Module B to improve its reliability.

III. TOPIC THREE: Using the Schneidewind Reliability Model

A. Failure Data Background: Data collection must be started at the development phases of the process including any failure data obtained from the developer-run tests. Data obtained from these early stages can then be used during the independent verification and validation phases to predict the software's reliability. However, this data collection would not stop at the development phase; data should be collected throughout field operations. Data obtained at this stage can be used for future software design projects and could lend itself to further model validation.

As discussed in the earlier sections of this training session, a model is only able to **make predictions** regarding the reliability of the software. These predictions can be used as a management aid for allocating resources and identifying the need for additional testing. They measure how reliable the software is compared to the desired reliability stated by management in the design specifications.

Modeling allows the manager to "get a feel" for how well the software performs based on actual data. This permits him to "look into the future" and predict how well the software will perform a week from now, a month from now, a year from now. . . The Schneidewind Model addresses the optimal selection of actual test data to be used in making software reliability predictions. The following sections describe the basic concepts used in this model and their implications for management. Numerous examples from the space *Shuttle* will be used because of the abundance of available test data . Where applicable, MCTSSA examples will additionally be discussed.

B. This section gives the manager additional information on the mathematical foundations of software reliability engineering. Application of the concepts discussed in the following lessons can be found in the Testing Methodologies Section starting on page 39 of the handbook. The following scenario is presented to give the reader an understanding of the model application and the uses for the application results:

- Time to Next Failure
- Remaining Failures and Fraction Remaining Failures
- Test Time to Achieve Specified Remaining Failures

- ♦ Cumulative Failures for a Specified Time
- ♦ Total Failures over the Life of the Software

C. Concepts Used in the Schneidewind Reliability Model

i. Time to Next Failure

a. Rationale: This concept is important for the manager in that it permits him to make an informed, educated decision on the reliability of the software. As a simplistic example, if the predicted time to next failure is three days, but the software is scheduled to be run for ten days, the manager can anticipate that a failure will occur before the mission is complete. He must then decide whether or not he wants to take that risk.

b. Concept Discussion: The time to next failure can be described as the amount of time that will elapse from the present time, t, until the next recorded failure occurs. In other words, it is the predicted amount of time it will take for the next failure to occur. Execution time is measured from the beginning of a test. This execution time is recorded in convenient intervals of time.

c. Application: As an example, a convenient interval of time for the Shuttle program is 30 days. This will be seen on the graphs displaying calculations of time to next failure. However, an organization can set its own interval. In some MCTSSA examples, an appropriate interval would be one week (five workdays).

Figure 3 (in the handbook, page 20) is a tool that can be used as a management aid. It shows the predicted and actual times to next failure for current execution times. The graph can be read in the following way. If we take a given failure, Failure 1, for example, it occurs at $t = 4$ (read from the x-axis); therefore, at $t = 1$, the time to next failure will be equal to 3 (read from the y-axis), $(4 - 1 = 3)$. At $t = 2$, the time to next failure will be equal to 2, $(4 - 2 = 2)$. At $t = 4$, Failure 1 occurs, so the time to next failure is 4, (8 - 4 = 4). In this figure, we **predict** the time to next failure to be 4 at execution time 18 on the **dashed curve**. This curve is derived from additional information and testing (using the Schneidewind Model).

ii. Remaining Failures and Fraction of Remaining Failures

a. Rationale: The number of remaining failures provides the manager with valuable information about the reliability of his software. Specifically, it gives him an indication of the software's reliability by predicting the remaining failures (undiscovered failures) that still exist in the software. With this information, he can make an informed decision as to whether the software meets his requirements. If the number of remaining failures is high, the software will typically not satisfy the reliability requirements.

b. Concept Discussion: The number of remaining failures, R, is measured from a given interval and identifies the *predicted count of failures remaining* in the software. If one predicts the total number of failures that will occur in the software, the remaining failures can be predicted though simple subtraction: total number of failures minus the number of failures found to date.

c. Application: Management will set guidelines on the desired value for R. Normally, R is set to be less than one (< 1). This means that the *expected* number of remaining failures that will occur from the present time to the end of the software execution cycle will be less than one. If the predicted value for R is greater than one, the software can be expected to fail during the mission. If the system is a mission critical or has the potential to cause harm to human life, the prediction of R >1 should tell the manager that there would be serious risk if he uses the software as it is currently designed. In figure 4, (in handbook, page 27), one can see how *p* (fraction of remaining failures) might behave as increased *test time* is applied (represented by "test intervals"). From this type of information a program manager can determine whether more testing is warranted, or whether the software is sufficiently tested to allow its release or unrestricted use. Note that required test time rises very rapidly at small values of **p** and **R(t)**.

iii. Test Time to Achieve Specified Remaining Failures

a. Rationale: For planning purposes, the manager can predict the total test time that would be required to achieve a given reliability level, as measured by number of remaining failures. The predicted total test time required to achieve a specified number of remaining failures, where $R(t_2)$ is the specified number of remaining failures at t_2, is:

$$t_2=[\log[\alpha/(\beta[R(t_2)])]]/\beta+(s-1)$$

b. Concept Discussion: An important trade-off is between reliability and the test time to achieve that reliability. As testing continues, the amount of test time required to achieve marginal increases in reliability become significant.

c. Application: Again we consult figure 4, page 27 in the handbook but this time focus on how rapidly test time increases with decreases in fraction remaining failures.

4. Cumulative Failures for a Specified Time

a. Rationale: It is useful to predict the cumulative failure count at future intervals -- before the defects are found -- so that the software manager can anticipate the reliability of the software and take *early* action to improve it, if necessary. Furthermore, management needs this information to plan tests and to assign personnel to testing and defect correction.

b. Concept Discussion: This gives the manager information about the reliability of the software from the start of testing or operation up to that time interval. It gives him information about the effectiveness of his quality assurance program and whether there is the need for additional testing. This information can prompt the manager to continue testing or to deploy the software, provided that the predicted time to next failure and number of remaining failures are acceptable.

c. Application: Cumulative failures are the total failures predicted to occur at a specific point of time in the future. The benefit of this prediction is that it can be used to anticipate the total failures, for a given execution time, and help the manager prepare to deal with them. Also, if the predicted number of failures is considered unacceptable, the software and its processes can be investigated to see where the problems lie.

5. Total Failures over the Life of the Software

a. Rationale: This quantity is the summation the failures predicted over the expected lifetime of the product. It can be used by management as an approximation of the "reliability" of the software under investigation and can be used as a measure of the product's reliability. Intuitively, a predicted large number of failures indicates poor reliability, with the converse also being true.

b. Concept Discussion: This quantity represents the total failures and faults in the software system. Therefore it is very useful for indicating the total testing problem that it confronts management in faces in order to achieve its reliability objectives.

c. Application: The main application is in the computation of remaining failures by subtracting the observed failures to date from the predicted total failures. For example, this approach was used in Figure 7, page 33 of the handbook in computing the ".6" remaining failures, which corresponds to a total test time of 52 intervals.

IV. **TOPIC FOUR: Software Reliability Prediction Tutorials**

1. **SMERFS and The Schneidewind Software Reliability Model**: SMERFS is a software reliability modeling tool that can be used to gain insight into the reliability of the software being tested. SMERFS is a tool that implements the models developed by Schneidewind and a number of other software reliability researchers. Using the Schneidewind Model component of SMERFS, two types of predictions can be made: for a given number of time intervals, how many failures will occur? secondly, for a given number of failures, how many time intervals will be required for the failures to occur? After inputing the software failure count data, usually from an input failure data file, the first step is to determine the optimal starting value for "s" as determined by the table of mean square error (MSE) values; usually the "s" with the minimum MSE will be selected.

2. **SMERFS Operations**:

   a. Although most of the directions for SMERFS show up on the computer screen and are self-explanatory, the following amplifying instructions will assist the first time user of SMERFS in successfully completing his session. See Appendix A to follow along with the SMERFS printout. User inputs are highlighted (in bold print) for ease of use. ***Note***: Calculation results should be rounded to no more than one or two decimal places, because reliability cannot be predicted with greater precision. However, to be consistent with the SMERFS printouts in Appendix A, the results shown in this section will be left as calculated.

   b. Once SMERFS is accessed, the first input required from the user will be the name of the file where he would like the SMERFS output (results) stored. As an example, **a:\smerfs1** would store the resulting SMERFS ASCII file on the computer's A-drive if a disk is inserted. This will make data retrieval easier once the session is complete. The user can then access his "output" file via a word processing program, format the data as he wishes, and print the results.

c. The user will then be asked if he would like to store a plot file for later retrieval. The recommended answer for this question is **0**, (zero), meaning "no".

d. SMERFS will next require the data type the user will be working with. At this point the user will enter **4**, for the interval failure counts and testing lengths.

e. Now he will be asked to enter **a one** for the standard SMERFS file input. This should be followed by the name of the file where his sample data is stored, for example, a file name of **oi618.in**. [This sample file contains the number of failures recorded against an operational increment (OI) of the *Shuttle*. The OI consists of a build of various modules in the *Shuttle* software library. There are 18 count intervals in **oi618.in**. Each interval is 30 days of continuous execution time.]

f. This step will ask the user how he would like the input displayed. The recommended response is to **enter a 3**. This entry will show a table of all the data input through the oi618.in file. However, the user may enter a 0 to display a list of his options at this point.

g. Following the display of data, the same question will reappear regarding the input display. This time the recommended response is to **enter 4** to take him to the SMERFS main menu. He will then be asked if he would like to make some new data files. He should **enter a 0** to void the data restore option.

h. He should then **enter 0** to display the listings available at the main menu. This will present him with nine choices. He should select **option 8 (Executions of the models).**

i. Upon this selection, the user will then **enter a 0** to display the available count model options. He should select **option 4 (The Schneidewind Model)**.

j. The next displays will permit the user to see descriptions of the model or the treatment type. For these options, a **0** should be entered unless he desires the descriptions.

k. The next step will be to investigate the "optimum s" from the various count intervals input into the program. A **1** should be entered here. He will then be asked to enter the range over which "s" should be tested. In general, the user should enter the range of the input failure data (i.e., 1,18 for this application). However for this application, we had previously determined that SMERFS could not compute values for MSE for "s" greater than 9. Therefore for this specific example, the user should **enter 1,9**. This entry will display the table of s, beta, alpha, WLS, MSE_F and MSE_T. The last two terms are the mean square error, as a function of "s", for number of failures and time to failure predictions, respectively (ignore the "WLS" column).

l. The user should note the table results and select those values for "s" which give him the **smallest** MSE_F and MSE_T.

m. After the user is comfortable with the data presented in the table, he should enter 0 to conclude the table presentation. He will then be asked to enter the desired model treatment number. He should enter **2**. For the number of associated values of "s" he should enter the corresponding "s" value that gives the smallest MSE_T, for time to failure prediction. In this sample file, the minimum value for MSE_T is seen for "s" equal to 5. A **5** should be entered. This entry will result in a display of model estimates. Of note in this display should be the *total number of failures,* and the *remaining number of failures.* (Total number of failures: .11722E+02; Remaining number of failures: .17221E+01). These values, as discussed previously, provide the manager with information regarding the reliability of the software he is testing. **He should record these values** for future use in this example.

3. TIME TO NEXT FAILURE PREDICTIONS

a. The user will then be prompted to select from two options regarding future predictions. For the sample run, he should select 2 for the prediction of the number of periods needed to discover the next "M" failures. This will allow him to determine the value of "M". He should **enter a 1**. The result will predict the number of *additional* test periods required to discover one more failure. The result is 6.34 periods (190 days). This implies that the time to next failure, from the present time, will be 190 days.

b. The user will be prompted to **enter a 0** to end the current predictions.

4. **NUMBER OF FAILURES PREDICTIONS**

a. This step moves the user into predicting the number of failures that will occur in one more test period. He will be prompted to enter the model treatment number. He should **enter a 2**.

b. He will then be prompted to enter the associated value of "s" he would like to investigate. He should enter the "s" value corresponding to the minimum value for MSE_F he recorded earlier. For this exercise, the **value of 6** should be entered. This entry will produce a listing similar to the listing produced in step 2k. As in step 2k, the key values obtained here are the *total number of failures*, the number corresponding to *plus those skipped*, and the *number of failures remaining*. If the value for *plus those skipped* is not equal to zero, this value must be added to the total number of failures and the number of failures remaining. **The user should record these values**. The example values correspond to:

Total number of failures:	14.363
Plus those skipped:	3
# of failures remaining:	4.3626

c. The program will present the user with two options for data evaluation. He should **choose option 1** for the number of failures expected in the next testing period. He will be prompted to enter the number of periods to examine. He should **enter a 1**. This will display the number of failures expected. For this example, it will be .369. This implies that the number of remaining failures occurring in the next operational increment (30 days) will be .36888. This is the final SMERFS calculation.

d. The user can exit the program by entering the following values in sequence: 0 to end predictions, followed by a 4 to terminate the model execution, 0 to conclude analysis of model fit, 0 for count model options, 6 to return to the main menu, 0 for a list of main module options, and finally, 9 to stop execution of SMERFS.

5. INTERPRETING SMERFS RESULTS

Using the sample file and the SMERFS software, the following results were achieved:

a. **Time to Failure Data** ("s" = 5):

Time to next failure (from present time): 6.34 periods (190 days)
Number of remaining failures (from present time): 1.72
Total number of failures: 11.7
Calculated *fraction of remaining failures*: .147

b. **Number of Failures Data** ("s" = 6):

Number of remaining failures (from present time): 7.36
Total number of failures: 17.36
Calculated *fraction of remaining failures*: .42
Predicted number of failures that will occur in one more period: .369

Note: Because in this example s=5 was optimal for *time to failure* predictions and s=6 was optimal for *number of failures* predictions, different results are obtained for *number of remaining failures* and *total number of failures*. Because MSE_F applies to failure count quantities like these, the values obtained for s=6 should be used in this example (i.e., *number of remaining failures*=7.36 and *total number of failures*=17.36).

These results provide the manager with useful information regarding the reliability of his software, provided he looks at all the data as complementary information. He should not make a decision based on only one piece of the above information, rather, he needs to look at the data in its entirety.

6. STATGRAPHICS OPERATIONS USING SHUTTLE DATA

a. Statgraphics is a statistical analysis program that is used to augment the reliability predictions obtained from SMERFS. Equations, like the one for t_2 below, can be created using the Statgraphics equation editor feature. Of particular interest in this phase of the predictions is the formula for computing the test time required to achieve a given reliability level. The amount of test time is defined by the following equation:

$$t_2 = [\log[\alpha/(\beta[R(t_2)])]]/\beta + (s-1)$$

b. Based on the way this equation is implemented in Statgraphics, the user must first calculate *p*, the fraction of remaining failures, for each of the desired number of remaining failures, $R(t_2)$. For this example, $R(t_2)$ will be one, two, three, and four.

c. Once Statgraphics has been accessed, the user will be presented with a menu showing various options for calculations and presentations. He will **depress the F8** function key which will cause a new screen to be superimposed on the menu. Here, he will type "**exec**" for the execution screens to appear.

d. Once the blank screens appear, he should **type t2** at the colon prompt if the user wants to see the equation before he uses it in a calculation. Otherwise, he can skip this step. This will display the above t_2 equation which has already been preloaded for the user. For Statgraphics to calculate the numerical value for this equation, the user must input the values for alpha, beta, Xs, s and p. The alpha, beta, and s values correspond to the values obtained from the SMERFS session for the smallest MSE_F value. The Xs value is the number of failures observed prior to s=6 from the same SMERFS session ("plus those skipped" in SMERFS); the p value is the desired number for the fraction of remaining failures for remaining failures of one, two, three, and four.

(1) The user will now enter the above mentioned values in the following format for one remaining failure:

```
 alpha GETS .73825
 beta GETS .051401
 Xs GETS 3
s GETS 6
p GETS (1/(EVAL Ft))
EVAL t2
 p
```

These commands will display the value for the test time required to achieve a reliability level. For this input, the predicted test time required to achieve the reliability level of having one remaining failure is 56.8 thirty day intervals. This will correspond to a fraction of remaining failures equal to .058. For the remaining failures equal to two, three, and four the following commands must be entered:

```
(2)     p GETS (2/(EVAL Ft))
        EVAL t2                          Results:  t2 is 43.35
        p                                          p is .115


(3)     p GETS (3/(EVAL Ft))
        EVAL t2                          Results:  t2 is 35.47
        p                                          p is .173

(4)     p GETS (4/(EVAL Ft))
        EVAL t2                          Results:  t2 is 29.87
        p                                          p is .230
```

The above results could be plotted to compare the effect that changing the remaining failures has on the amount of test time needed to achieve that end. An asymptotic relationship is seen between t2 and the fraction of remaining failures, "p." Figure 10 (in handbook, page 49) is a sample graph that could be obtained.

e. Application: With this information, the manager could gain insight into the predicted amount of time it would take to achieve given reliability levels.

V. TOPIC FIVE: Applications of Prediction Results

1. Interpreting SMERFS Results

Using the sample file and the SMERFS software, the following results were obtained:

a. **Time to Failure Data** ("s" = 5):

Time to next failure (from present time): 6.34 periods (190 days)
Number of remaining failures (from present time): 1.72
Total number of failures: 11.7
Calculated *fraction of remaining failures*: .147

b. **Number of Failures Data** ("s" = 6):

Number of remaining failures (from present time): 7.36
Total number of failures: 17.36
Calculated *fraction of remaining failures*: .42
Predicted number of failures that will occur in one more period: .369

These results provide the manager with useful information regarding the reliability of his software, provided he looks at all the data as complementary information. He should not make a decision based on only one piece of the above information, rather, he needs to look at the data in its entirety.

c.. Application of Results: A manager must decide whether or not to launch the space *Shuttle* for a mission to last ten days. He has collected failure data on the software to be used in the launch and has input the data into the model as described in the above sections. Based on his confidence in the model and the predictions made by the model he will make his decision to launch or not. With the above statistical predictions at hand, the manager could make a decision on whether or not to launch the *Shuttle* for a mission lasting ten days. Looking at the data in its entirety, he should **not launch** the *Shuttle*. Even though the time to next failure is predicted at 190 days and only .37 failures are predicted for the next interval (30 days giving the mission a cushion of 20 days), the predicted number of remaining failures is 7.36. This is a significantly high number. (As discussed previously, the manager desires this number to be less than one.) The time to make the software virtually error free is 72 periods, a long period of time. The decision must be based on the available model evidence, his confidence in the model, his risk aversion, and any other factors at his disposal. Using only the data from this experiment, the overriding factor of 7.36 possibly life-threatening failures, the manager should not launch the *Shuttle*.

2. Interpreting Statgraphics Results

a. Application: With this information, the manager could gain insight into the predicted amount of time it would take to achieve a given reliability level. Using the scenario mentioned previously, as an example, one could see that it is predicted to take almost 57 periods (totaling 4.7 years) from t=0 to reduce the fraction of remaining failures to .058. The test time curve, Figure 10, page 49 in the handbook, indicates that there will be a point where there are only marginal returns achieved by additional testing.

Looking at the shape of the curve, the software manager must understand that as predicted reliability increases (the number of predicted failures decreases) there will be a significant increase in the amount of testing time needed to achieve those results. There will come a point were the additional cost of testing will result in only minimal gains in reduced software failures. The manager must make the decision to stop testing and deploy the current software, based on available funding for testing and the desired reliability levels.

Management must use all resources available to it to come to a sound, information-supported decision. The predictions provided by the Schneidewind Software Reliability Model can give management additional information on the predicted reliability of its software. This can be accomplished by both the developer and the implementor using the software reliability engineering process that has been described in the handbook. Using appropriate data, the predictions can be used to help make an informed reliability decision. However, the final decision must be made by the manager based on all the information he has available to him.

APPENDIX A: Sample Training Session (Held at MCTSSA, 14 December 1995)

ASTERISKS INDICATE COMMENTS TO DISTINGUISH THEM FROM SMERFS OUTPUT

READ IN DATA THAT WAS PREVIOUSLY GENERATED BY SMERFS FROM ASCII FILE INPUT

TESTING INTERVAL WILL ALWAYS BE "1" IN SMERFS. IN THE APPLICATION IT WILL BE THE ACTUAL LENGTH OF EACH INTERVAL (E.G., 1 HOUR, 1 DAY, 30 DAYS)

```
 ENTER DESIRED DATA TYPE, OR ZERO FOR A LIST.
     4  *REFERS TO FAILURE COUNT DATA*

 ENTER ONE FOR A STANDARD SMERFS FILE INPUT; ELSE ZERO.
     1

 ENTER INPUT FILE NAME FOR INTERVAL DATA.
oi618.in    *(Shuttle Failure Data)*
 THE INPUT OF  18 INTERVAL ELEMENTS WAS PERFORMED.

 ENTER INPUT OPTION, OR ZERO FOR A LIST.
     0
 THE AVAILABLE INPUT OPTIONS ARE:
  1 ASCII FILE INPUT
  2 KEYBOARD INPUT
  3 LIST THE CURRENT DATA
  4 RETURN TO THE MAIN PROGRAM
 ENTER INPUT OPTION.
     3

INTERVAL  NO. OF FAULTS    TESTING LENGTH
=======   ===========     ============
   1   .00000000E+00   .10000000E+01
   2   .00000000E+00   .10000000E+01
   3   .00000000E+00   .10000000E+01
   4   .00000000E+00   .10000000E+01
   5   .30000000E+01   .10000000E+01
   6   .10000000E+01   .10000000E+01
   7   .00000000E+00   .10000000E+01
   8   .10000000E+01   .10000000E+01
   9   .00000000E+00   .10000000E+01
  10   .10000000E+01   .10000000E+01
  11   .10000000E+01   .10000000E+01
  12   .00000000E+00   .10000000E+01
  13   .20000000E+01   .10000000E+01
  14   .00000000E+00   .10000000E+01
  15   .00000000E+00   .10000000E+01
  16   .00000000E+00   .10000000E+01
  17   .00000000E+00   .10000000E+01
  18   .10000000E+01   .10000000E+01
```

FIND THE BEST STARTING INTERVAL FOR USING THE FAILURE DATA. SINCE THERE IS A TOTAL OF 18 INTERVALS OF DATA, USE THE RANGE 1,18. SMERFS WILL ONLY PRODUCE A RESULT FOR "S" WHERE IT CAN OBTAIN CONVERGENCE. FOR FAILURE COUNT PREDICTIONS, USE THE MINIMUM MSE-F "S"; FOR TIME TO FAILURE PREDICTIONS, USE THE MINIMUM MSE-T "S".

```
ENTER ONE TO INVESTIGATE FOR THE OPTIMUM S (USING TREATMENT TYPE
  NUMBER 2); ELSE ZERO TO CONTINUE WITH THE MODEL EXECUTION.
     1
  ENTER RANGE OVER WHICH S  SHOULD BE TESTED.  NOTE, AN EXECUTION
  ON A GIVEN S WHICH FAILED THE CONVERGENCE CRITERIA  WILL NOT BE
  INCLUDED IN THE FOLLOWING RESULTS TABLE.  THE OPTIMUM S FOR EI-
  THER MSE-F OR MSE-T IS THE ONE  RESULTING IN THE SMALLEST VALUE
  FOR YOUR CHOSEN CRITERIA.
     1      18

  S  BETA        ALPHA       WLS         MSE-F       MSE-T
  == =========== =========== =========== =========== ===========
   1 .37154E-02 .57434E+00 .71189E+00 .89573E+00 .15098E+01
   2 .25076E-01 .72250E+00 .84899E+00 .68418E+00 .12947E+01
   3 .52370E-01 .92300E+00 .10130E+01 .47735E+00 .10803E+01
   4 .88195E-01 .12021E+01 .12214E+01 .34612E+00 .86076E+00
   5 .13700E+00 .16059E+01 .15409E+01 .47758E+00 .60788E+00
   6 .51401E-01 .73825E+00 .58125E+00 .24450E+00 .11042E+01
   7 .28025E-01 .58878E+00 .50090E+00 .30476E+00 .13863E+01
   9 .60985E-01 .66786E+00 .61535E+00 .28068E+00 .13683E+01

   ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL  EXECUTION.
     2          *METHOD WHEREBY INTERVALS 1,..., S-1 ARE DISCARDED*

  ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).
     6          *CORRESPONDS TO MINIMUM MSE-F ABOVE BECAUSE WE WILL BE MAKING A
FAILURE COUNT  PREDICTION.*

 TREATMENT 2 MODEL ESTIMATES ARE:
  BETA                  .51401E-01
  ALPHA                 .73825E+00
  TOTAL NUMBER OF FAULTS    .14363E+02
   PLUS THOSE SKIPPED       .30000E+01 IN PERIODS 1 THROUGH   5 *(INTERVALS 1,...,S-1)*
  # OF FAULTS REMAINING     .43626E+01
  WEIGHTED SUMS-OF-SQUARES
   BETWEEN PREDICTED AND
   OBSERVED FAULTS          .58125E+00
  MEAN SQUARE ERROR FOR
   CUMULATIVE FAULTS        .24450E+00
  MEAN SQUARE ERROR FOR
   TIME TO NEXT FAILURE     .11042E+01
```

*** CORRECT PREDICTED TOTAL NUMBER OF FAILURES = 14.36+3.0 (NUMBER SKIPPED)=17.36***

*** ACTUAL TOTAL NUMBER OF FAILURES=14 (FAILURES OBSERVED AFTER 65.03 INTERVALS)***

*** CORRECT PREDICTED NUMBER OF REMAINING FAILURES=4.26+3.0 (NUMBER SKIPPED)=7.36***

```
*ACTUAL NUMBER OF REMAINING FAILURES=4 (FAILURES OBSERVED BETWEEN 18 AND 65.03
INTERVALS)*

 THE AVAILABLE FUTURE PREDICTIONS ARE:

  1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD

  2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
 ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

       1

  ENTER NUMBER OF PERIODS TO EXAMINE, OR ZERO TO END.

    1.000000000000000            *WANT PREDICTION FOR INTERVAL T=19*

 # OF FAULTS EXPECTED     .36888E+00

  *ACTUAL NUMBER OF FAILURES IN NEXT INTERVAL (T=19)=0*

  *MODEL IS ENTERED AGAIN SO THAT THE BEST VALUE OF "S" FOR TIME TO FAILURE
PREDICTION CAN   BE USED*

 ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL   EXECUTION.

       2

 ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).

      5          *CORRESPONDS TO MINIMUM MSE-T ABOVE BECAUSE WE WILL BE MAKING A
TIME TO FAILURE  PREDICTION.*

*THE USUAL LISTING IS NOT SHOWN BECAUSE THE TOTAL FAILURES AND REMAINING FAILURES
WERE OBTAINED AS PART OF THE FAILURE COUNT PREDICTION*

 THE AVAILABLE FUTURE PREDICTIONS ARE:

  1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD

  2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
 ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

       2

  ENTER VALUE OF M (BETWEEN ONE AND  .17221E+01), OR ZERO TO END. *(THIS IS THE RANGE OF
    PREDICTED REMAINING FAILURES)*

    1.000000000000000            *WANT PREDICTION FOR ONE MORE FAILURE*

  # OF PERIODS EXPECTED     .63443E+01 *(I.E., T=18+6.34=24.34)*

*ACTUAL TIME TO NEXT FAILURE=6.2 (I.E., T=18+6.2=24.2)*
```

```
 ENTER INPUT FILE NAME FOR INTERVAL DATA.

logais30.in          *LOGAIS FILE FOR 30 INTERVALS (SUBMIT DAY)*
                     *SEE TABLE 3, PAGES 78-84* IN THE HANDBOOK*

 THE INPUT OF  30 INTERVAL ELEMENTS WAS PERFORMED.

 ENTER INPUT OPTION, OR ZERO FOR A LIST.

      0

  THE AVAILABLE INPUT OPTIONS ARE:

  1 ASCII FILE INPUT
  2 KEYBOARD INPUT
  3 LIST THE CURRENT DATA
  4 RETURN TO THE MAIN PROGRAM
  ENTER INPUT OPTION.

      3

INTERVAL  NO. OF FAULTS      TESTING LENGTH
========  =============      ==============

    1    .12000000E+03     .10000000E+01
    2    .18500000E+03     .10000000E+01
    3    .10000000E+01     .10000000E+01
    4    .19100000E+03     .10000000E+01
    5    .21300000E+03     .10000000E+01
    6    .17800000E+03     .10000000E+01
    7    .54000000E+02     .10000000E+01
    8    .39000000E+02     .10000000E+01
    9    .15000000E+02     .10000000E+01
   10     .28000000E+02     .10000000E+01
   11     .99000000E+02     .10000000E+01
   12     .70000000E+02     .10000000E+01
   13     .60000000E+02     .10000000E+01
   14     .10000000E+02     .10000000E+01
   15     .10500000E+03     .10000000E+01
   16     .11500000E+03     .10000000E+01
   17     .82000000E+02     .10000000E+01
   18     .59000000E+02     .10000000E+01
   19     .73000000E+02     .10000000E+01
   20     .60000000E+01     .10000000E+01
   21     .18000000E+02     .10000000E+01
   22     .19000000E+02     .10000000E+01
   23     .32000000E+02     .10000000E+01
   24     .31000000E+02     .10000000E+01
   25     .20000000E+02     .10000000E+01
   26     .70000000E+01     .10000000E+01
   27     .10000000E+02     .10000000E+01
   28     .21000000E+02     .10000000E+01
   29     .54000000E+02     .10000000E+01
   30     .14000000E+02     .10000000E+01

 ENTER ONE TO INVESTIGATE FOR THE OPTIMUM S (USING TREATMENT TYPE
 NUMBER 2); ELSE ZERO TO CONTINUE WITH THE MODEL EXECUTION.
      1
  ENTER RANGE OVER WHICH S  SHOULD BE TESTED.  NOTE, AN EXECUTION
  ON A GIVEN S WHICH FAILED THE CONVERGENCE CRITERIA  WILL NOT BE
  INCLUDED IN THE FOLLOWING RESULTS TABLE.  THE OPTIMUM S FOR EI-
  THER MSE-F OR MSE-T IS THE ONE  RESULTING IN THE SMALLEST VALUE
  FOR YOUR CHOSEN CRITERIA.
      1      30

  S   BETA          ALPHA         WLS           MSE-F       MSE-T
 ==== =========== =========== =========== =========== ===========
   1 .69434E-01  .15299E+03  .42946E+04  .31693E+04  .85711E+00
   2 .72231E-01  .14901E+03  .42323E+04  .33143E+04  .84940E+00
  23 .25195E-02  .23864E+02  .20573E+03  .16798E+03  .32778E+00

 ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL
 EXECUTION.

      2
  ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).

      23

  TREATMENT 2 MODEL ESTIMATES ARE:
  BETA                  .25195E-02
  ALPHA                  .23864E+02
  TOTAL NUMBER OF FAULTS    .94715E+04
   PLUS THOSE SKIPPED      .17400E+04 IN PERIODS 1 THROUGH  22
  # OF FAULTS REMAINING    .75425E+04
  WEIGHTED SUMS-OF-SQUARES
   BETWEEN PREDICTED AND
   OBSERVED FAULTS         .20573E+03
  MEAN SQUARE ERROR FOR
   CUMULATIVE FAULTS        .16798E+03
  MEAN SQUARE ERROR FOR
   TIME TO NEXT FAILURE     .32778E+00
```

*** CORRECT PREDICTED TOTAL NUMBER OF DEFECTS = 9471.5+1740.0 (NUMBER SKIPPED)=11211.5***

*** ACTUAL TOTAL NUMBER OF DEFECTS=? (WON'T KNOW UNTIL LOGAIS RETIRED FROM SERVICE)***

*** CORRECT PREDICTED NUMBER OF REMAINING DEFECTS=7542.5+1740.0 (NUMBER SKIPPED)= 9282.5***

ACTUAL NUMBER OF REMAINING DEFECTS=? (WON'T KNOW UNTIL LOGAIS RETIRED FROM SERVICE)

```
THE AVAILABLE FUTURE PREDICTIONS ARE:

 1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
 2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS

 ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.

     2
```

(See Figure 16, page 59 in the handbook for plots of the following results)*

```
 ENTER VALUE OF M (BETWEEN ONE AND  .75425E+04), OR ZERO TO END.

   56.000000000000000          *NUMBER OF OBSERVED DEFECTS IN THE RANGE 31,35 (DAYS)
                                SEE LOGAIS DATA*

 # OF PERIODS EXPECTED     .24017E+01       *ACTUAL=5*

 ENTER VALUE OF M (BETWEEN ONE AND  .75425E+04), OR ZERO TO END.

   164.000000000000000         *NUMBER OF OBSERVED DEFECTS IN THE RANGE 31,40 (DAYS)
                                SEE LOGAIS DATA*

 # OF PERIODS EXPECTED     .70749E+01       *ACTUAL=10*

 ENTER VALUE OF M (BETWEEN ONE AND  .75425E+04), OR ZERO TO END.

   433.000000000000000         *NUMBER OF OBSERVED DEFECTS IN THE RANGE 31,45 (DAYS)*

 # OF PERIODS EXPECTED     .18960E+02       *ACTUAL=15*

 ENTER VALUE OF M (BETWEEN ONE AND  .75425E+04), OR ZERO TO END.

   495.000000000000000         *NUMBER OF OBSERVED DEFECTS IN THE RANGE 31,50 (DAYS)*

 # OF PERIODS EXPECTED     .21750E+02       *ACTUAL=20*

 ENTER VALUE OF M (BETWEEN ONE AND  .75425E+04), OR ZERO TO END.

   987.000000000000000         *NUMBER OF OBSERVED DEFECTS IN THE RANGE 31,55 (DAYS)*

 # OF PERIODS EXPECTED     .44618E+02       *ACTUAL=25*
```

APPENDIX C. LOGAIS DEFECT DATA

Table 3: *LOGAIS* Chronological Defect Counts

Count Interval (*t*)	Defect ID Range	Number of Defects	Summit Date	Day
1	1-120	120	11/11/94	Fri
2	121-305	185	11/12/94	Sat
3	306	1	11/13/94	Sun
4	307-497	191	11/14/94	Mon
5	498-710	213	11/15/94	Tue
	5 Day Total	**710**		
	Cumulative	**710**		
6	711-888	178	11/16/94	Wed
7	889-942	54	11/17/94	Thu
8	943-981	39	11/18/94	Fri
9	982-996	15	11/19/94	Sat
10	997-1024	28	11/20/94	Sun
	5 Day Total	**314**		
	Cumulative	**1024**		
11	1025-1123	99	11/21/94	Mon
12	1124-1193	70	11/22/94	Tue
13	1194-1253	60	11/23/94	**Wed**
14	1254-1263	10	11/25/94	**Fri**
15	1264-1368	105	11/28/94	**Mon**
	5 Day Total	**344**		
	Cumulative	**1368**		
16	1369-1483	115	11/29/94	Tue
17	1484-1565	82	11/30/94	Wed
18	1566-1624	59	12/1/94	Thu
19	1625-1697	73	12/2/94	Fri

20	1698-1703	6	12/3/94	Sat
	5 Day Total	**335**		
	Cumulative	**1703**		
21	1704-1721	18	12/4/94	Sun
22	1722-1740	19	12/5/94	Mon
23	1741-1772	32	12/6/94	Tue
24	1773-1803	31	12/7/94	Wed
25	1804-1823	20	12/8/94	Thu
	5 Day Total	**120**		
	Cumulative	**1823**		
26	1824-1830	7	**12/9/94**	**Fri**
27	1831-1840	10	**12/15/94**	**Thu**
28	1841-1861	21	**12/19/94**	**Mon**
29	1862-1915	54	12/20/94	Tue
30	1916-1929	14	12/21/94	Wed
	5 Day Total	**106**		
	Cumulative	**1929**		
31	1930-1935	6	12/22/94	Thu
32	1936-1960	25	**12/23/94**	**Fri**
33	1961-1964	4	**12/28/94**	**Wed**
34	1965-1982	18	12/29/94	Thu
35	1983-1985	3	**12/30/94**	**Fri**
	5 Day Total	**56**		
	Cumulative	**1985**		
36	1986	1	**1/3/95**	**Tue**
37	1987-2000	14	1/4/95	Wed
38	2001-2003	3	1/5/95	Thu
39	2004-2027	24	**1/6/95**	**Fri**
40	2028-2093	66	1/9/95	Mon

	5 Day Total	**108**		
	Cumulative	**2093**		
41	2094-2157	64	1/10/95	Tue
42	2158-2231	74	1/11/95	Wed
43	2232-2292	61	1/12/95	Thu
44	2293-2358	66	1/13/95	Fri
45	2359-2362	4	**1/14/95**	**Sat**
	5 Day Total	**269**		
	Cumulative	**2362**		
46	2363-2372	10	**1/16/95**	**Mon**
47	2373-2390	18	1/17/95	Tue
48	2391-2399	9	1/18/95	Wed
49	2400-2405	6	1/19/95	Thu
50	2406-2424	19	**1/20/95**	**Fri**
	5 Day Total	**62**		
	Cumulative	**2424**		
51	2425-***	48	**1/24/95**	**Tue**
52	2426-***	44	1/25/95	Wed
53	2430-***	145	1/26/95	Thu
54	2433-***	227	**1/27/95**	**Fri**
55	2446-2473	28	**1/30/95**	**Mon**
	5 Day Total	**492**		
	Cumulative	**2916**		
56	2474-2480	7	1/31/95	Tue
57	2481-2486	6	2/1/95	Wed
58	2487-2510	24	2/2/95	Thu
59	2511-2529	19	**2/3/95**	**Fri**
60	2530-2543	14	2/6/95	Mon
	5 Day Total	**70**		

	Cumulative	**2986**		
61	2544***	53	2/7/95	Tue
62	3040-3067	28	2/8/95	Wed
63	3068-3099	32	2/9/95	Thu
64	3100-3110	11	**2/10/95**	**Fri**
65	3111-3137	27	2/13/95	Mon
	5 Day Total	**151**		
	Cumulative	**3137**		
66	3138-3146	9	2/14/95	Tue
67	3147-3167	21	2/15/95	Wed
68	3168-3213	46	2/16/95	Thu
69	3214-3233	20	**2/17/95**	**Fri**
70	3234-3242	9	**2/21/95**	**Tue**
	5 Day Total	**105**		
	Cumulative	**3242**		
71	3243-3260	18	2/22/95	Wed
72	3261-3314	54	2/23/95	Thu
73	3315-3320	6	**2/24/95**	**Fri**
74	3321-3324	4	**2/27/95**	**Mon**
75	3325-3334	10	2/28/95	Tue
	5 Day Total	**92**		
	Cumulative	**3334**		
76	3335-3340	6	3/1/95	Wed
77	3341	1	3/2/95	Thu
78	3342-3343	2	**3/3/95**	**Fri**
79	3344-3347	4	**3/6/95**	**Mon**
80	3348-3349	2	3/7/95	Tue
	5 Day Total	**15**		
	Cumulative	**3349**		
81	3350-3362	13	**3/8/95**	**Wed**
82	3363-3368	6	**3/10/95**	**Fri**

83	3369-3379	11	**3/13/95**	**Mon**
84	3380-3383	4	3/14/95	Tue
85	3384-3419	36	3/15/95	Wed
	5 Day Total	**70**		
	Cumulative	**3419**		
86	3420-3431	12	3/16/95	Thu
87	3432-3447	16	**3/17/95**	**Fri**
88	3448-3492	45	**3/20/95**	**Mon**
89	3493-3530	38	3/21/95	Tue
90	3531-3566	36	3/22/95	Wed
	5 Day Total	**147**		
	Cumulative	**3566**		
91	3567-3601	35	3/23/95	Thu
92	3602-3616	15	**3/24/95**	**Fri**
93	3617-3635	19	**3/27/95**	**Mon**
94	3636-3652	17	3/28/95	Tue
95	3653-3658	6	3/29/95	Wed
	5 Day Total	**92**		
	Cumulative	**3658**		
96	3659-3681	23	3/30/95	Thu
97	3682-3693	12	**3/31/95**	**Fri**
98	3694-3710	17	**4/3/95**	**Mon**
99	3711-3726	16	4/4/95	Tue
100	3727-3731	5	4/5/95	Wed
	5 Day Total	**73**		
	Cumulative	**3731**		
101	3732-3769	38	4/6/95	Thu
102	3770-3840	71	**4/7/95**	**Fri**
103	3841	1	**4/10/95**	**Mon**

104	3842-3856	15	4/11/95	Tue
105	3857-3885	29	4/12/95	Wed
	5 Day Total	**154**		
	Cumulative	**3885**		
106	3906-***	23	4/13/95	Thu
107	3909-3923	15	**4/14/95**	**Fri**
108	3924-3932	9	**4/16/95**	**Sun**
109	3933-3949	17	4/17/95	Mon
110	3950-3963	14	4/18/95	Tue
	5 Day Total	**78**		
	Cumulative	**3963**		
111	3964-4033	70	4/19/95	Wed
112	4034-4079	46	**4/20/95**	**Thu**
113	4080-4107	28	**4/25/95**	**Tue**
114	4108-4132	25	4/26/95	Wed
115	4133-4167	35	4/27/95	Thu
	5 Day Total	**204**		
	Cumulative	**4167**		
116	4168-4176	9	**4/28/95**	**Fri**
117	4177-4185	9	**5/1/95**	**Mon**
118	4186-4207	22	5/2/95	Tue
119	4208-4287	80	5/3/95	Wed
120	4288-4320	33	5/4/95	Thu
	5 Day Total	**153**		
	Cumulative	**4320**		
121	4321-4328	8	**5/5/95**	**Fri**
122	4329-4343	15	**5/8/95**	**Mon**
123	4344-4356	13	5/9/95	Tue
124	4357-4367	11	5/10/95	Wed

125	4368-4418	51	5/11/95	Thu
	5 Day Total	**98**		
	Cumulative	**4418**		
126	4419-4504	86	**5/12/95**	**Fri**
127	4505-4513	9	**5/15/95**	**Mon**
128	4514-4555	42	5/16/95	Tue
129	4556-4584	29	5/17/95	Wed
	TOTAL	**4584**		

*** Indicates discontinuous sequences of IDs for Submit Date. First ID of first sequence shown.

Bolding indicates breaks in defect submit dates.

LIST OF REFERENCES

Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

IEEE/AIAA Standard for a Software Quality Metrics Methodology, IEEE 1061 June, 1993.

Billings, Charles, "Journey to a Mature Software Process," *IBM Systems Journal,* Vol. 33, No. 1, 1994.

Farr, William and Oliver D. Smith, *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide*, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

Keller, Ted, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software," *Proceedings of the AIAA Computing in Aerospace 10*, San Antonio, TX, March 28, 1995.

Marine Air-Ground Task Force II/ Logistics Automated Information System, *Test Plan,* Summer, 1995.

Marine Air-Ground Task Force II, Logistics Automated Information System, *Logistics Systems Integration Review,* Technical Report Task 1, April 1, 1992

Marine Corps Tactical Systems Support Activity, "An Activity Overview of the Marine Corps Tactical Systems Support Activity, Camp Pendleton, CA," July 1994.

Marine Corps Tactical Systems Support Activity, Camp Pendleton, CA, "Statement of Work," prepared for Naval Postgraduate School, Monterey, CA dated February 2, 1995.

Musa, John, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability:Measurement, Prediction, Application*, McGraw-Hill, NY, 1987.

Schneidewind, Norman F., "Reliability and Risk Analysis for Software that Must be Safe," *Proceedings of the International Symposium on Software Metrics*, Berlin, Germany, March 25-26, 1996.

Schneidewind, Norman F. and Judie A. Heineman, "Technical Decision Paper," prepared for Marine Corps Tactical Systems Support Activity, Camp Pendleton, CA, April 15, 1995.

Schneidewind, Norman F. and Judie A. Heineman, "Technology Demonstration Report," prepared for Marine Corps Tactical Systems Support Activity, Camp Pendleton, CA, October 11 1995.

Schneidewind, Norman F., "Software Reliability Model with Optimal Selection of Failure Data," *IEEE Transactions on Software Engineering*, Vol. 19, No. 11, November 1993.

Schneidewind, Norman F.and Ted Keller, "Application of Reliability Models to the Space Shuttle," *IEEE Software*, Vol. 9, No. 4, July 1992.

Schneidewind, Norman F., "Analysis of Error Processes in Computer Software," *Proceedings of the International Conference on Reliable Software*, IEEE Computer Society, April 21-23, 1975.

Stark, George E., *Software Reliability Measurement Handbook*, June 1992.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3.	Professor Norman F. Schneidewind, Code SM/Ss Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Marine Corps Tactical Systems Support Activity Box 555171 Bldg. 31345 Camp Pendleton, CA 92055-5171	1
5.	LCDR Judie A. Heineman 194 North Henry Street Brooklyn, NY 11222	4